시각화와 분석

모듈 - 1

강사: 장순용 박사

순서

시각화 : 목적

시각화의 목적:

- 가설과 가설 사이의 비교 목적.
- 데이터의 인과 관계, 상응 관계, 구조적 관계를 보여주려는 목적.
- 다변량 데이터를 요약해서 보여주려는 목적.
- 분석 결과를 요약해서 보여주려는 목적.
- 분석의 결론을 뒷받침하는 증거물로 제시하려는 목적.

컨텐츠 (스토리)의 **유**/**무**가 시각화의 효과를 결정한다!

시각화 : 목적

탐색적 시각화의 목적:

- 데이터의 **통계적 특성**을 알아보려는 목적.
- 데이터에 패턴이 있는지 육안으로 확인해 보려는 목적.
- 향후 분석 방향을 정하려는 목적.
- 결과를 보고하려는 목적.

시각화 : 특징

탐색적 시각화의 특징:

- 분석가 본인의 직관적인 이해를 우선시 한다.
- 많은 노력을 들이지 않고 짧은 시간안에 완성한다.
- 많은 수의 그래프 생성한다.
- 타이틀, 레이블 등은 크게 중요하지 않다.
- 컬러, 폰트, 등은 꼭 필요할 때만 사용한다.

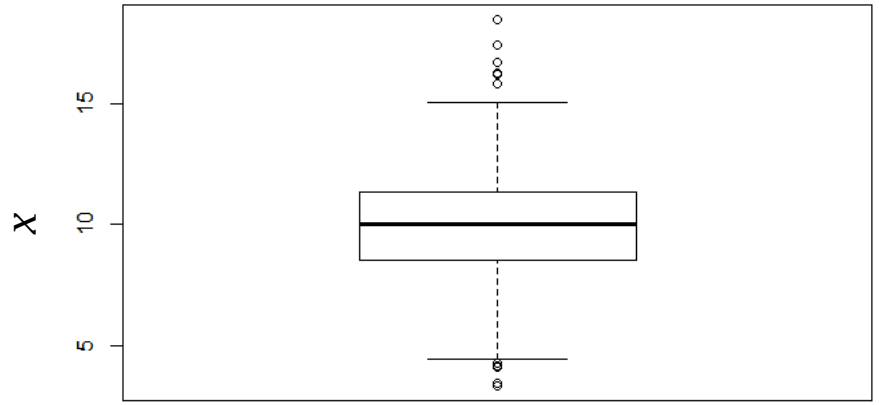
시각화 : 용도와 유형

시각화 용도와 유형:

- **단변량 기술통계 요약**: **상자그림(boxplot)**, 히스토그램(histogram), 막대그림(bar chart), 파이(pie), 등.

  → 연속형 변수 1개를 사용한다.

  → Box, Whiskers, Outliers로 구성된다.



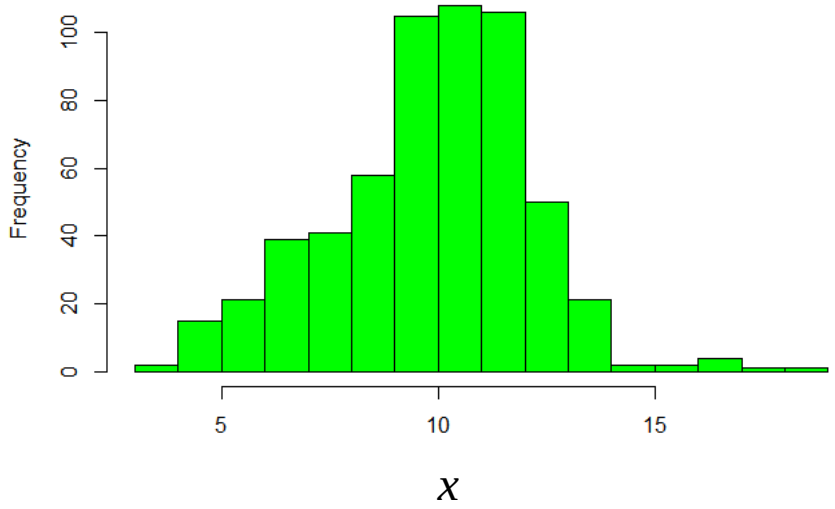
시각화 : 용도와 유형

시각화 용도와 유형:

- **단변량 기술통계 요약**: 상자그림(boxplot), **히스토그램(histogram)**, 막대그림(bar chart), 파이(pie), 등.

  → 연속형 변수 1개를 사용한다.

  → 각 계급에 해당하는 자료의 도수 (count)를 보여준다.

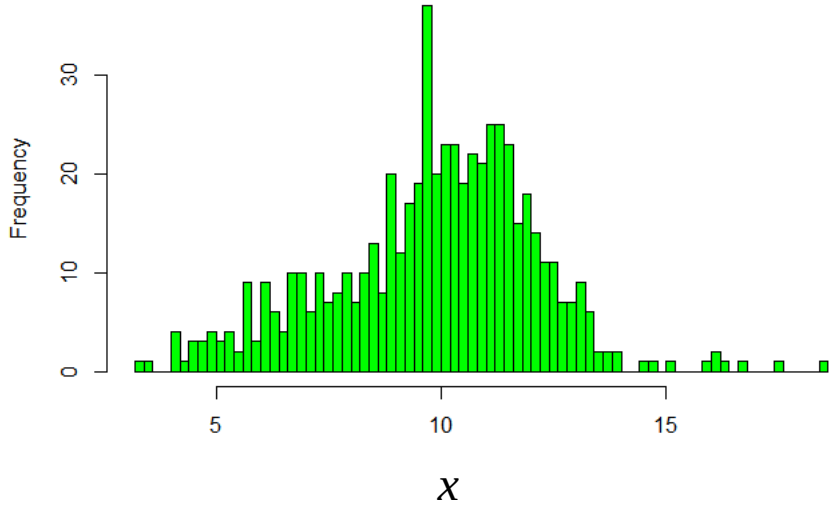


시각화 : 용도와 유형

시각화 용도와 유형:

- **단변량 기술통계 요약**: 상자그림(boxplot), **히스토그램(histogram)**, 막대그림(bar chart), 파이(pie), 등.

  → 계급의 크기를 조정해 줄 수 있다.



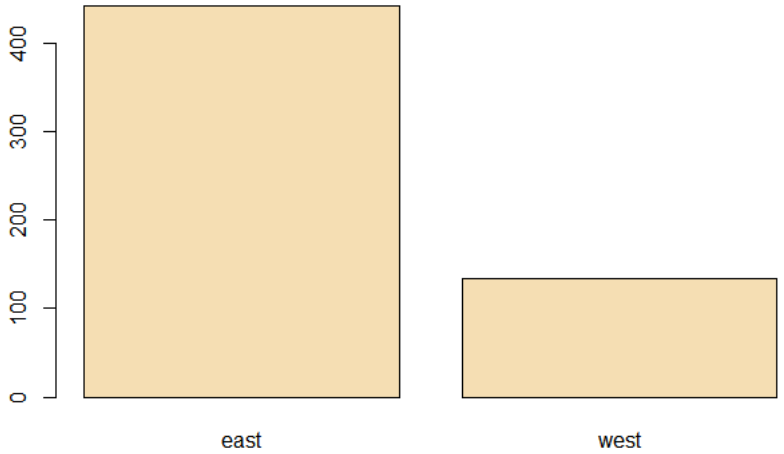
시각화 : 용도와 유형

시각화 용도와 유형:

- **단변량 기술통계 요약**: 상자그림(boxplot), 히스토그램(histogram), **막대그림(bar chart)**, 파이(pie), 등.

  → 명목형 변수 1개를 사용한다.

  → 명목형 변수 유형별 도수 (count)를 보여준다.



시각화 : 용도와 유형

시각화 용도와 유형:

- **단변량 기술통계 요약**: 상자그림(boxplot), 히스토그램(histogram), 막대그림(bar chart), **파이(pie)**, 등.

  → 명목형 변수 1개를 사용한다.

  → 명목형 변수 유형별 상대 도수 (비율)를 보여준다.

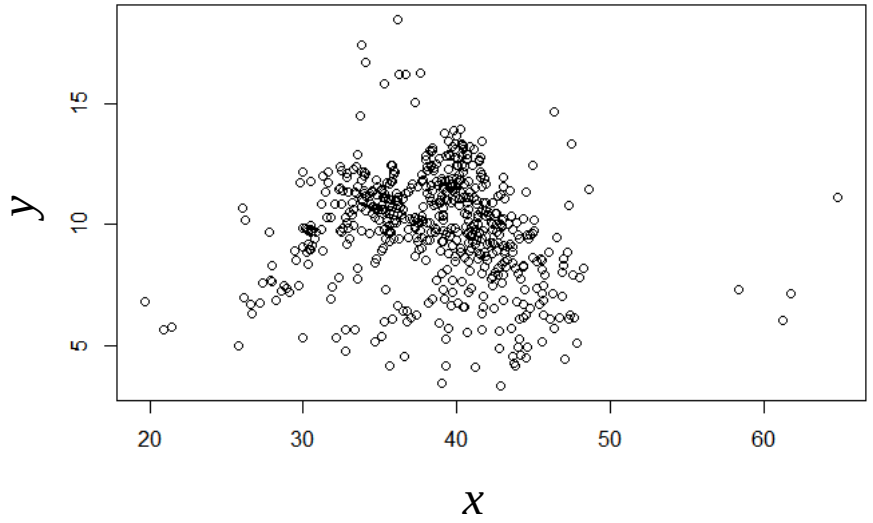
시각화 : 용도와 유형

시각화 용도와 유형:

- **비교, 상관관계 확인**: **산점도**, 다중 산점도, 다중 히스토그램, 다중 상자그림, 등.

  → 기본적으로 연속형 변수 2개를 사용한다: X와 Y.

  → 두개의 연속형 변수 사이의 관계를 보여준다: X = 독립, Y = 종속.

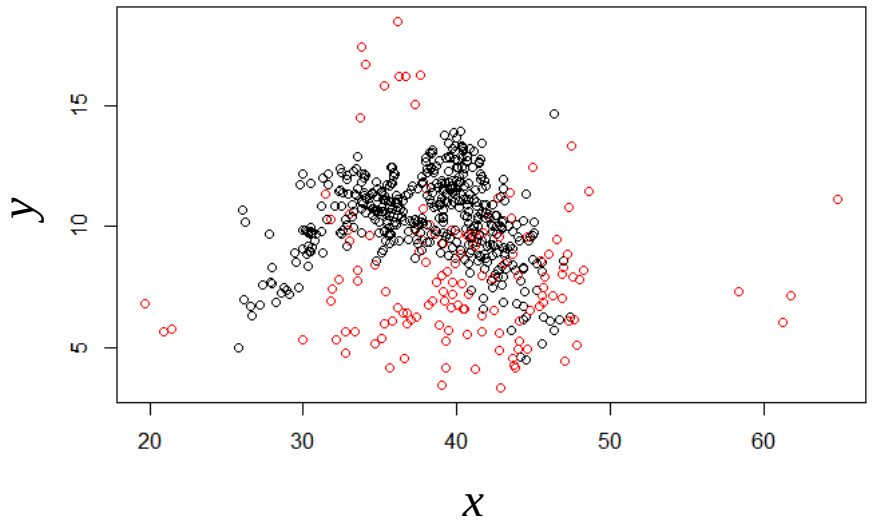


시각화 : 용도와 유형

시각화 용도와 유형:

- **비교, 상관관계 확인**: **산점도**, 다중 산점도, 다중 히스토그램, 다중 상자그림, 등.

  → 연속형 변수 2개 이외에 명목형 변수가 1개 있다면 색상으로 표현할 수 있다.



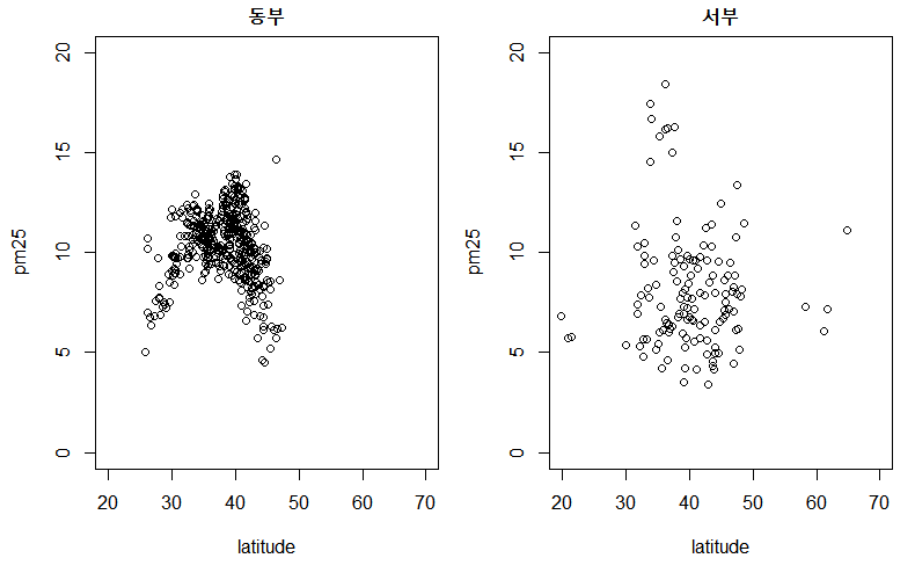
시각화 : 용도와 유형

시각화 용도와 유형:

- **비교, 상관관계 확인**: 산점도, **다중 산점도**, 다중 히스토그램, 다중 상자그림, 등.

  → 연속형 변수 2개 이외에 명목형 변수가 1개 있다면 여러 개의 산점도로 나누어서 표현할 수 있다.

  → 산점도의 수 = 명목형 변수 유형의 수.



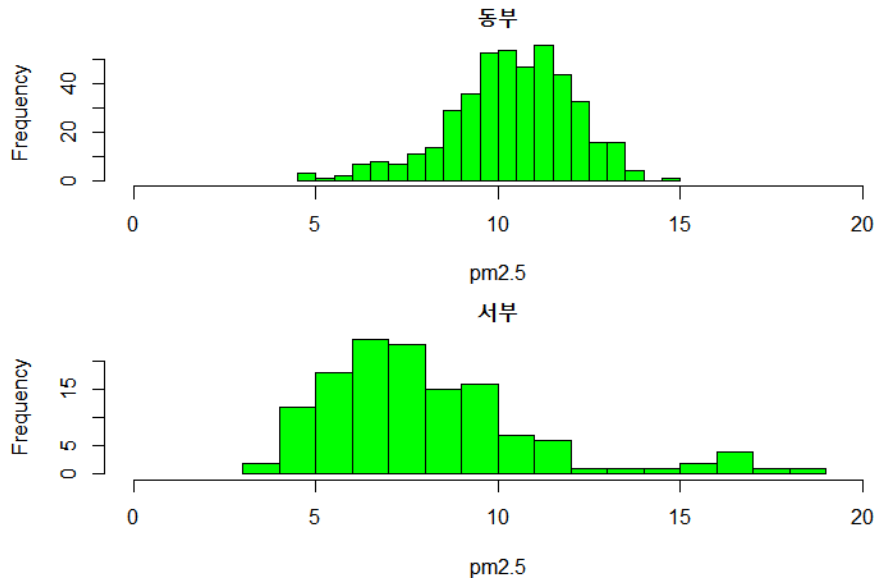
시각화 : 용도와 유형

시각화 용도와 유형:

- **비교, 상관관계 확인**: 산점도, 다중 산점도, **다중 히스토그램**, 다중 상자그림, 등.

  → 명목형 변수 1개 + 연속형 변수 1개를 사용한다.

  → 히스토그램의 수 = 명목형 변수 유형의 수.



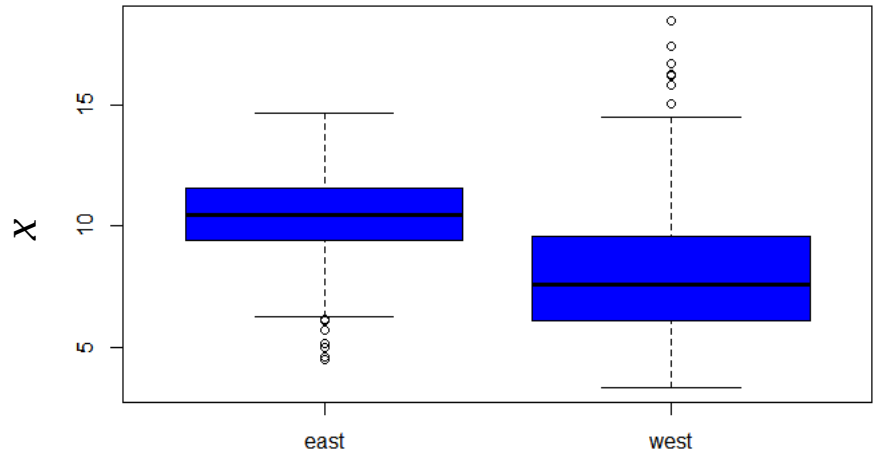
시각화 : 용도와 유형

시각화 용도와 유형:

- **비교, 상관관계 확인**: 산점도, 다중 산점도, 다중 히스토그램, **다중 상자그림**, 등.

  → 명목형 변수 1개 + 연속형 변수 1개를 사용한다.

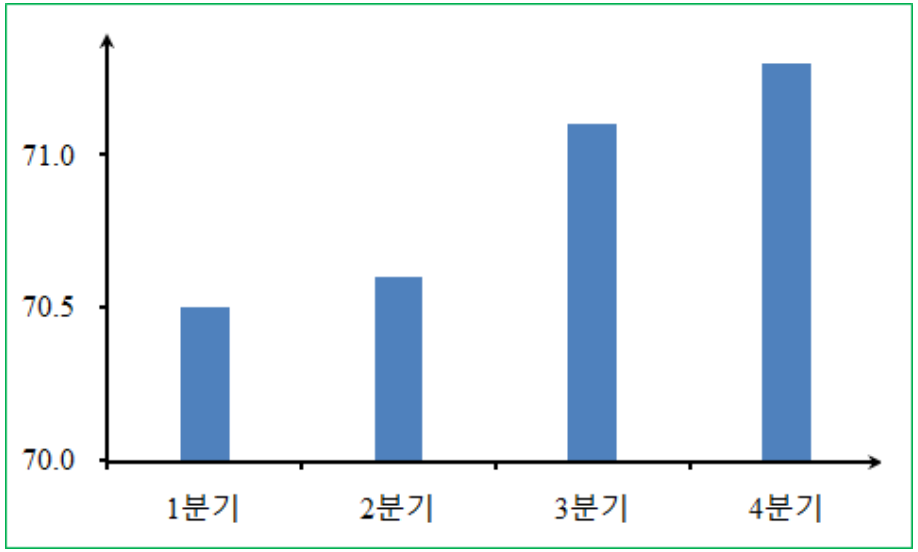
  → 상자 그림의 수 = 명목형 변수 유형의 수.



시각화 : 착시 현상

시각화와 착시현상:

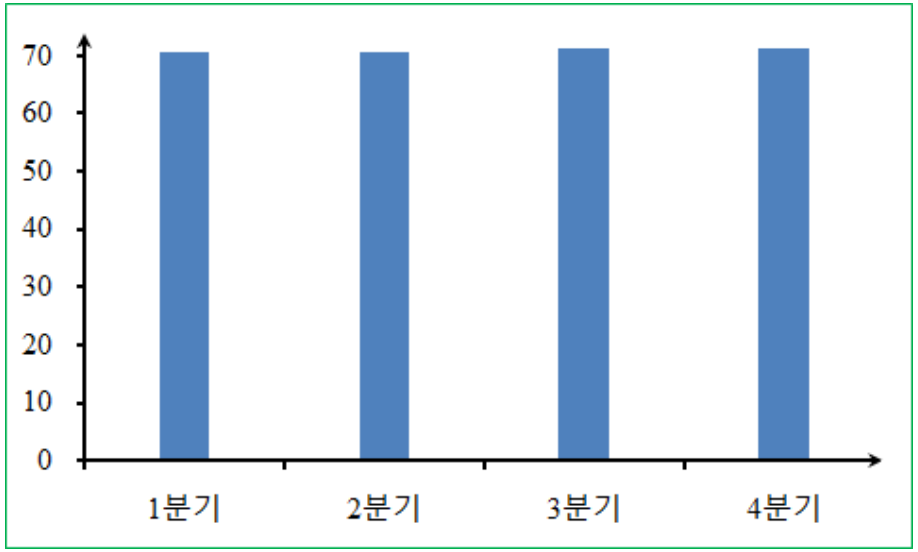
- A사의 분기별 매출 현황을 다음과 같이 막대그림으로 표현해 본다. ⇒ 호황?



시각화 : 착시 현상

시각화와 착시현상:

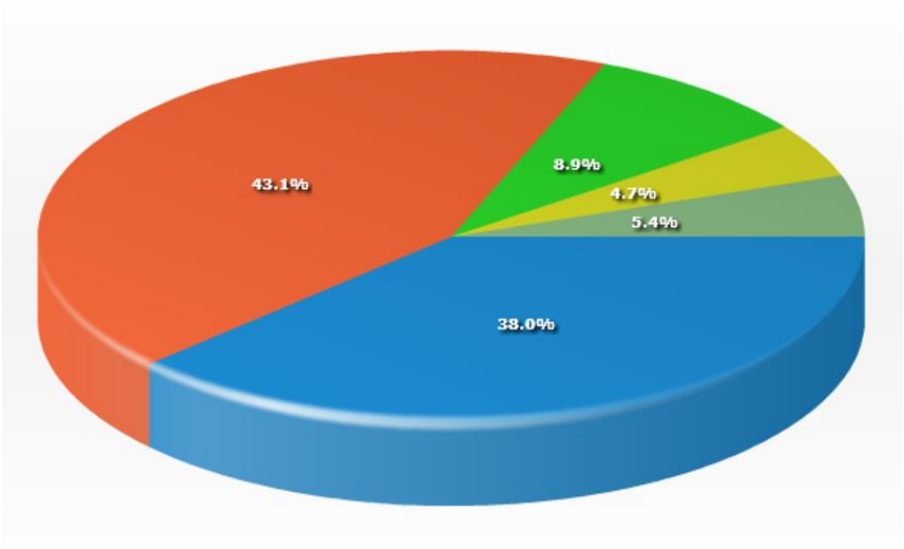
- 세로축 스케일을 조정하여 전체를 보여주도록 만든다. ⇒ 대략 2% 매출 증가!



시각화 : 착시 현상

시각화와 착시현상:

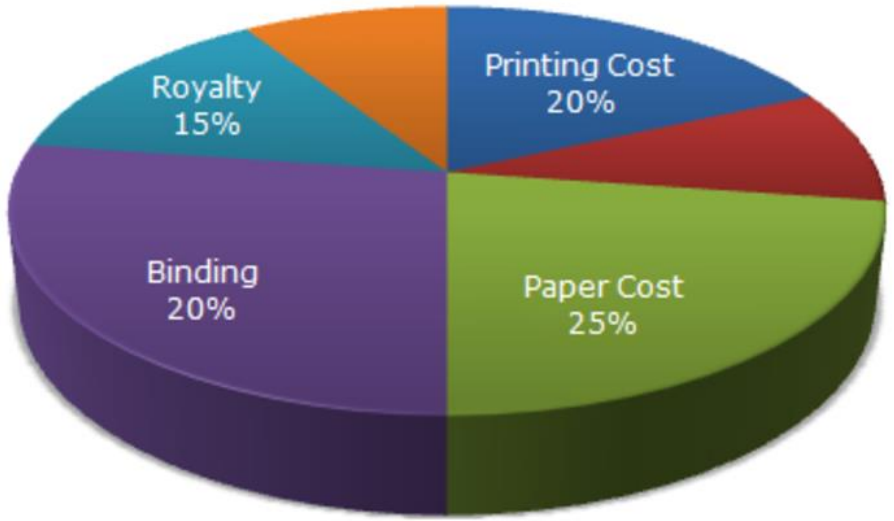
- 3D 파이차트에서는 상대적 비율을 분간하기 힘들다!
- 3D 파이차트 에서는 원근법 때문에 매우 큰 착시현상이 발생한다!



시각화 : 착시 현상

시각화와 착시현상:

- 3D 파이차트에서는 상대적 비율을 분간하기 힘들다!
- 3D 파이차트 에서는 원근법 때문에 매우 큰 착시현상이 발생한다!



순서

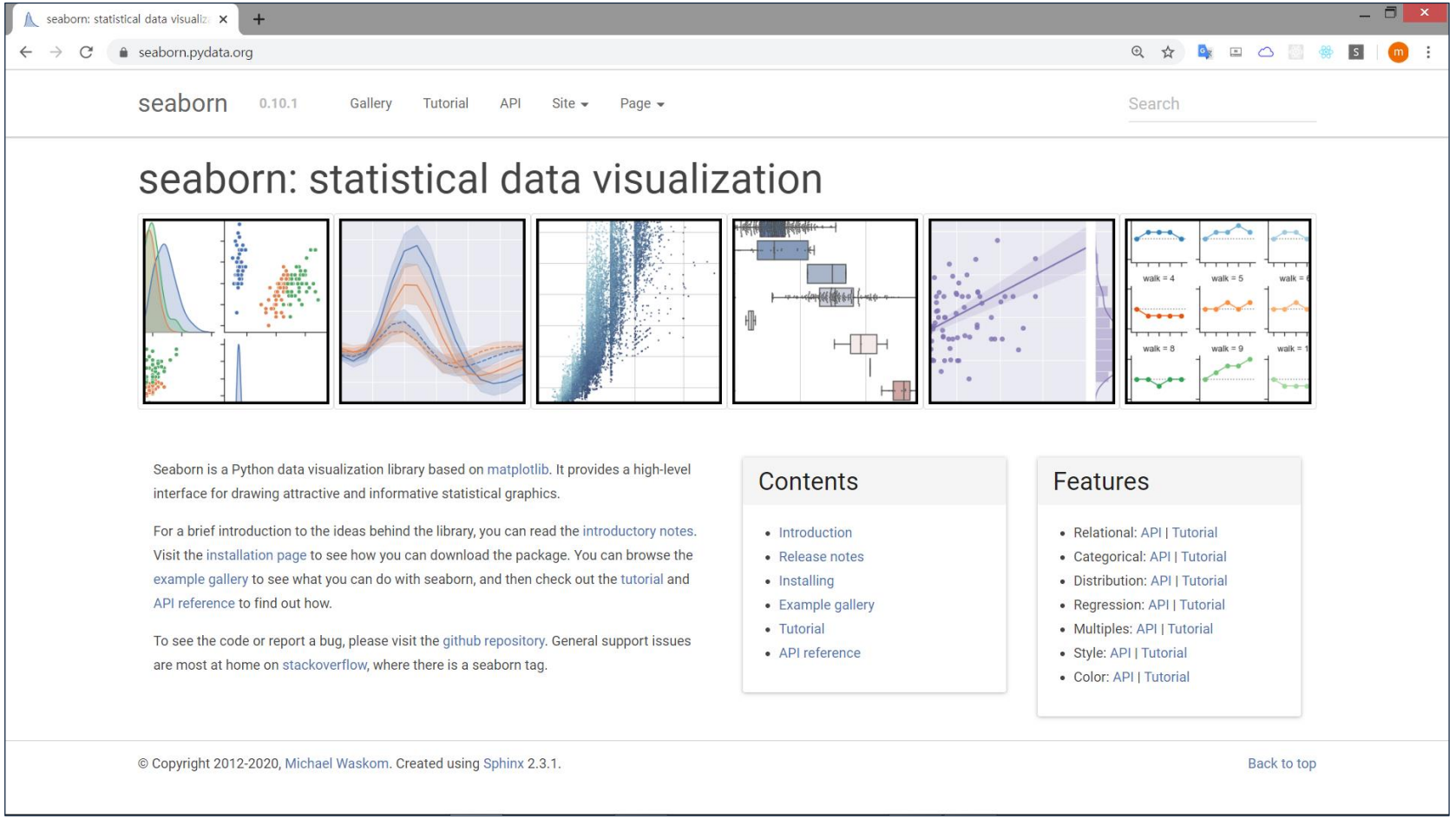
Seaborn 라이브러리 : 개요

Seaborn의 주요기능:

- 내장 데이터: load_dataset().
- 기본 시각화 유형: histplot(), displot(), scatterplot(), jointplot(), kdeplot(), boxplot(), barplot(), countplot(), 등.
- 시각화 행렬 유형: pairplot(), PairGrid(), FacetGrid(), 등.
- 회귀선 추가 기능: lmplot(), jointplot(), 등.
- 2D 특수 시각화: heatmap(), clustermap(), 등.
- 기본 시각화의 변형: violinplot(), swarmplot(), stripplot(), 등.

Seaborn 라이브러리 : 개요

관련 사이트: *https://seaborn.pydata.org*



Seaborn 라이브러리 : 개요

Seaborn의 내장 데이터: *https://github.com/mwaskom/seaborn-data*

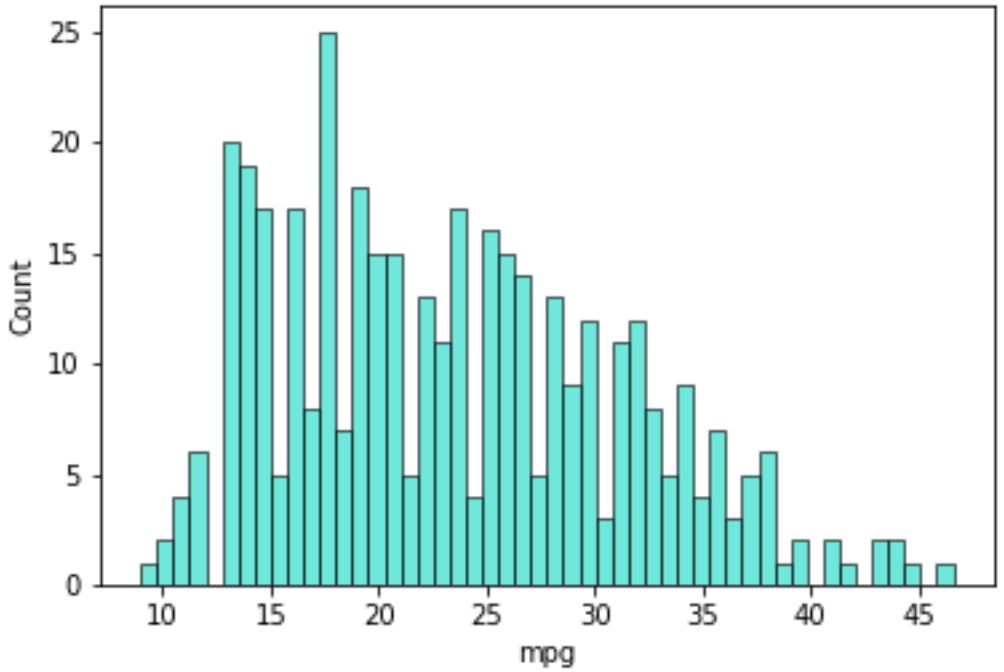
명칭	내용
mpg	자동차 연비 데이터.
iris	붓꽃 데이터.
titanic	타이태닉호 데이터.
diamonds	다이아몬드의 스펙과 가격 데이터.
car_crashes	자동차 사고 관련 데이터.
flights	1949~1960 사이의 항공기 운항 횟수 데이터.
geyser	“Old Faithful” 간헐천 분출 주기 데이터.

⇒ 제공되는 데이터셋 중 일부만 간추림.

Seaborn 라이브러리 : 히스토그램 #1

히스토그램:

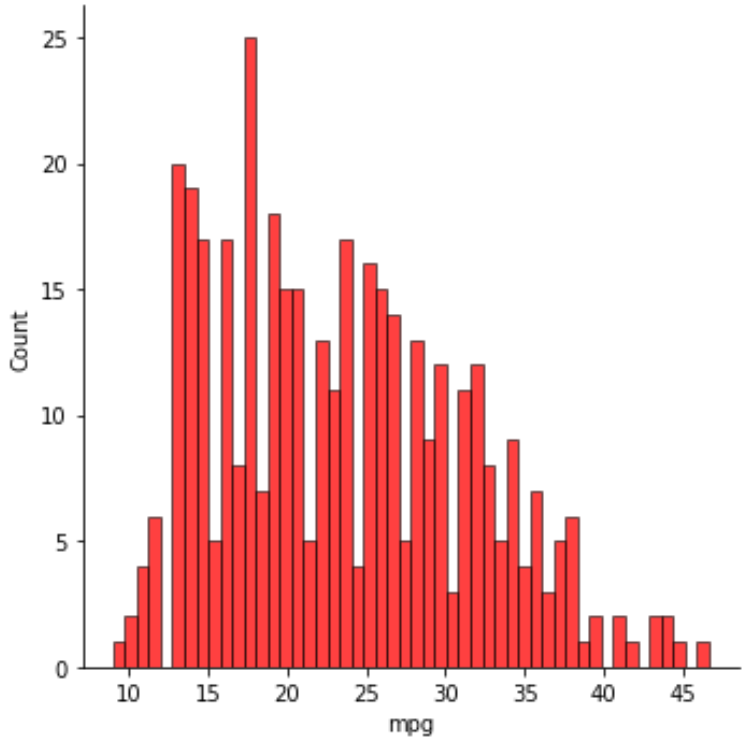
```
fig = sns.histplot(data=df, x='mpg',  bins=50, color='turquoise')
```



Seaborn 라이브러리 : 히스토그램 #2

히스토그램: 다른 방법.

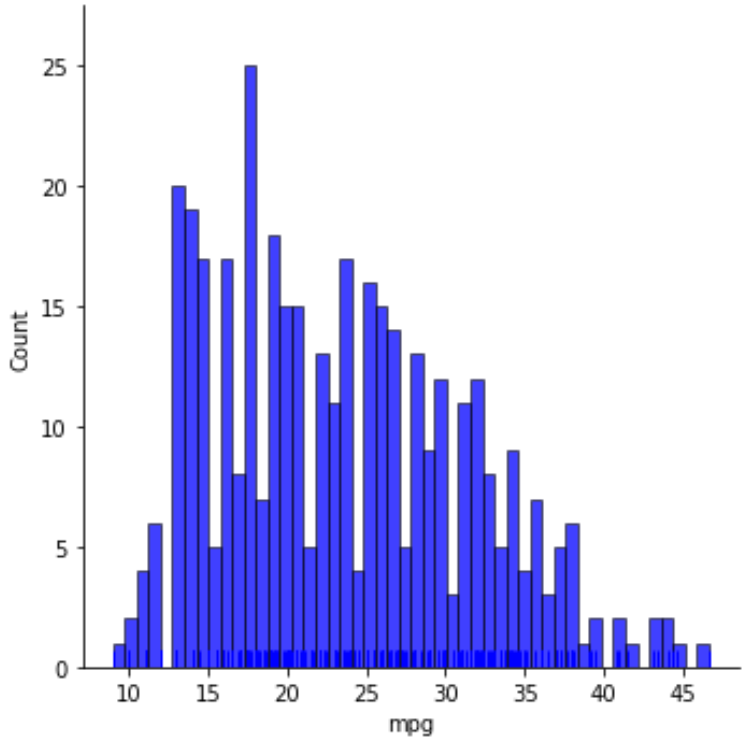
```
fig = sns.displot(data=df, x='mpg', kde=False, rug=False, bins=50, color='red', kind='hist' )
```



Seaborn 라이브러리 : 히스토그램 + Rug

히스토그램 + Rug:

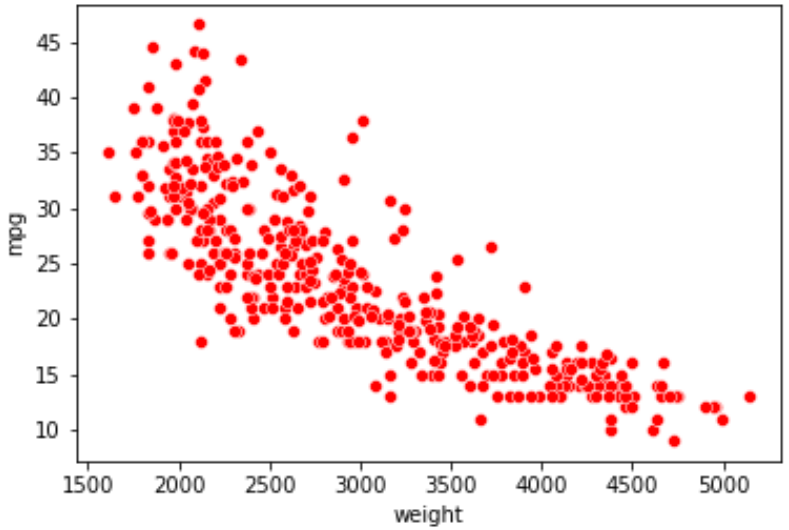
```
fig = sns.displot(data=df, x='mpg', kde=False, rug=True, bins=50, color='blue', kind='hist' )
```



Seaborn 라이브러리 : 산점도

산점도: 기본형.

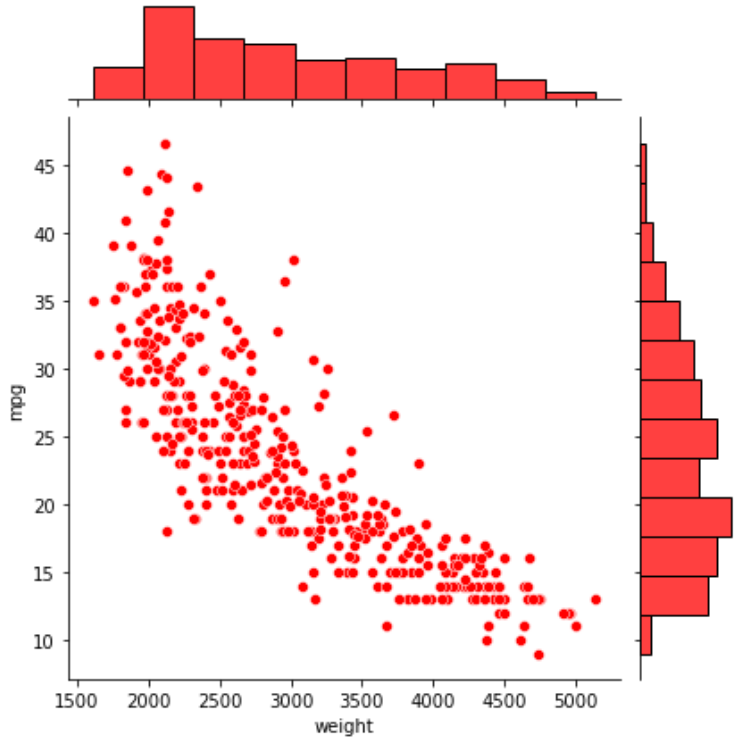
```
fig = sns.scatterplot(data=df, x='weight', y='mpg', color='red')
```



Seaborn 라이브러리 : 산점도

산점도: 다른 방법.

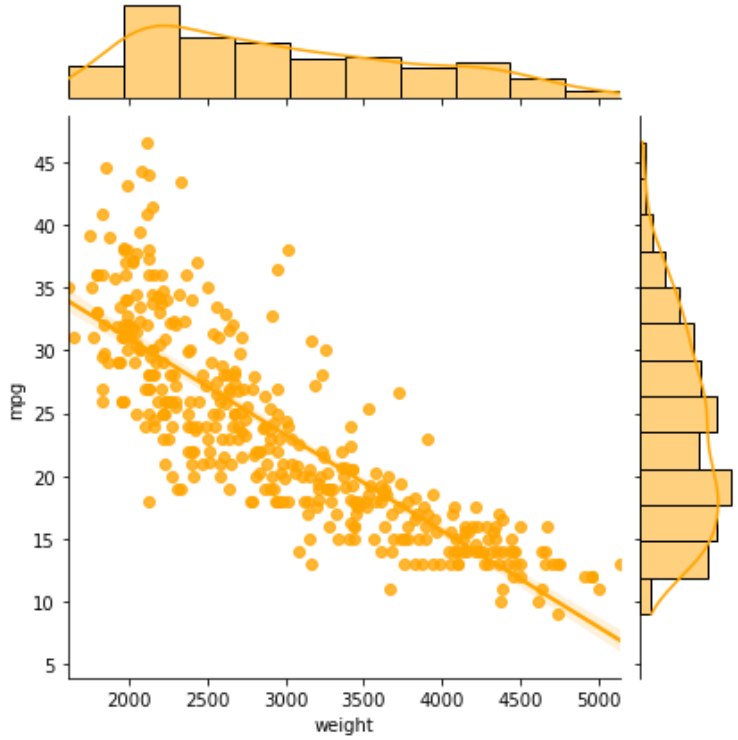
```
fig = sns.jointplot(data=df, x='weight', y='mpg', color='red', kind='scatter')
```



Seaborn 라이브러리 : 산점도 + 회귀선

산점도 + 회귀선:

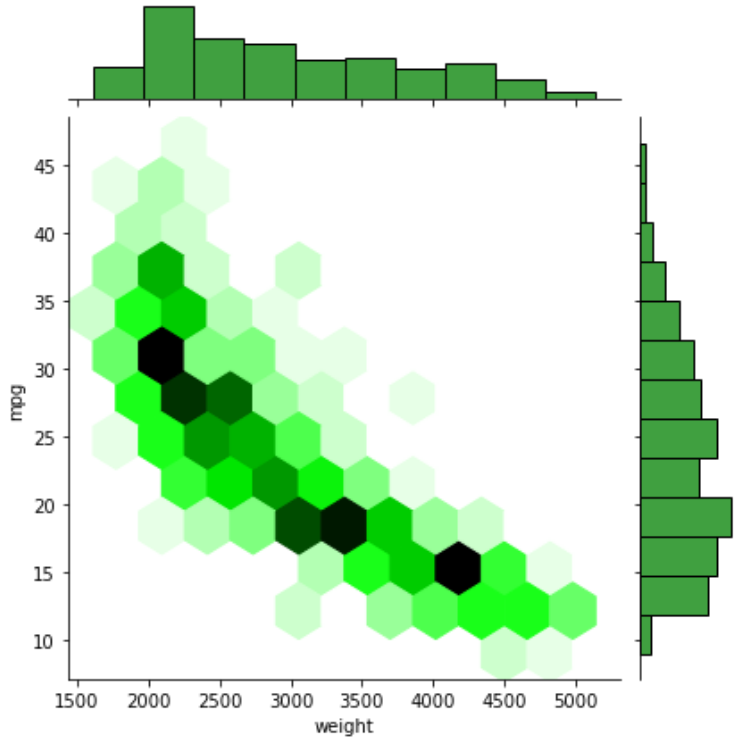
```
fig = sns.jointplot(data = df, x='weight', y='mpg', color='orange', kind='reg')
```



Seaborn 라이브러리 : Hex

Hex:

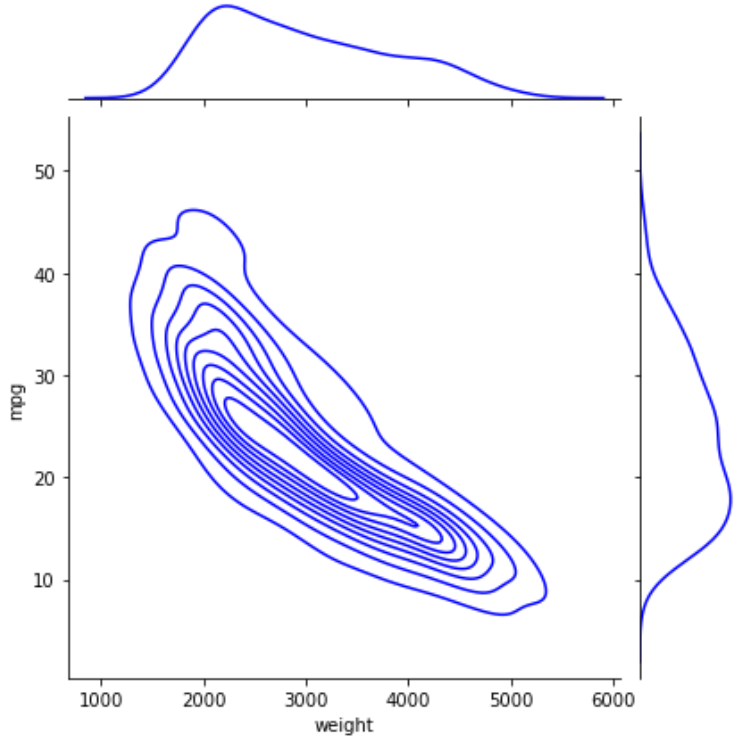
```
fig = sns.jointplot(data = df, x='weight', y='mpg', color='green', kind='hex')
```



Seaborn 라이브러리 : KDE

KDE (Kernel Density Estimation):

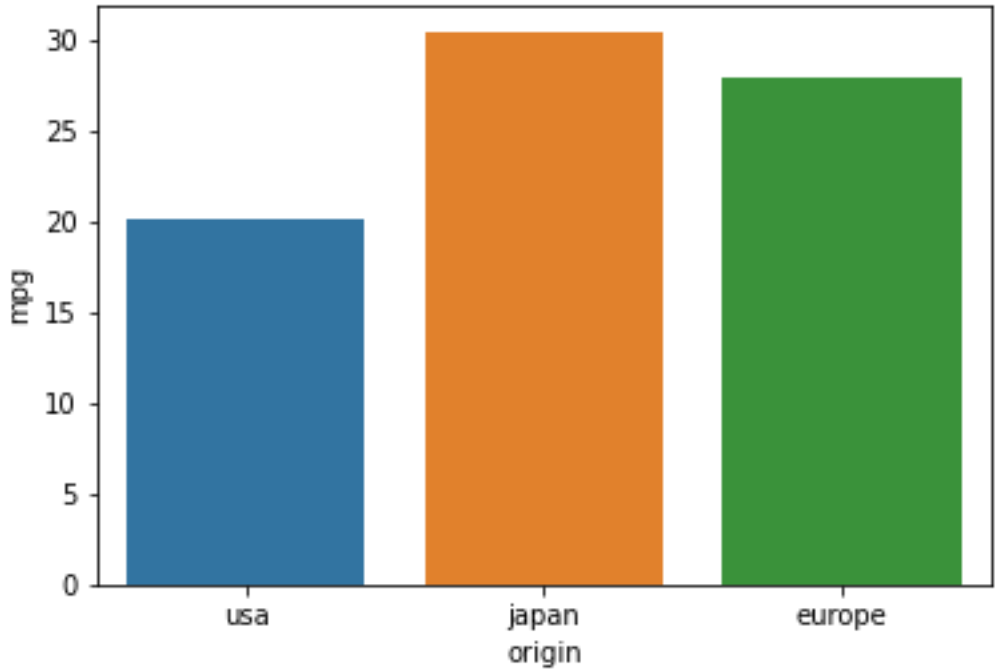
```
fig = sns.jointplot(data= df, x='weight', y='mpg', color='blue', kind='kde')
```



Seaborn 라이브러리 : 막대그림

막대그림: 유형별 집계.

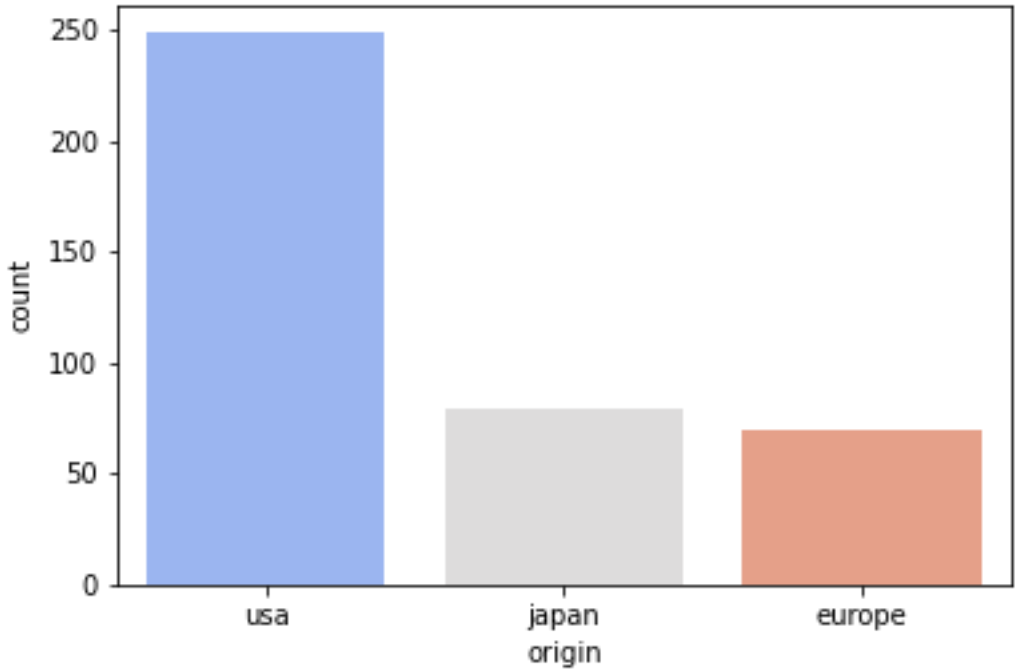
```
fig = sns.barplot(data=df, x='origin', y='mpg', ci=False)
```



Seaborn 라이브러리 : 막대그림

막대그림: 도수표.

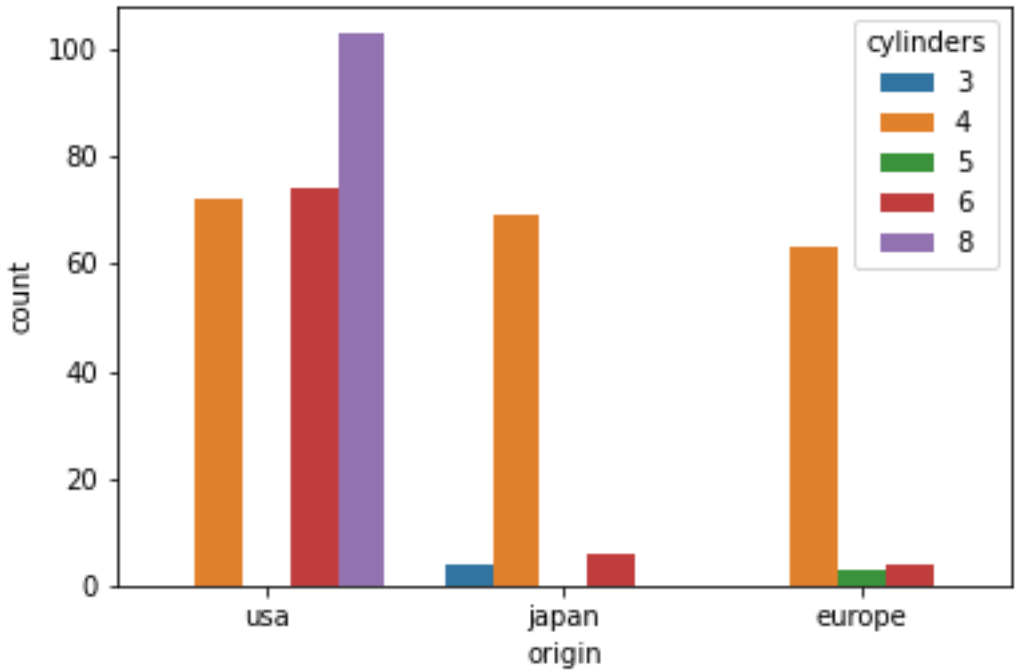
```
fig = sns.countplot(data=df, x='origin', palette ='coolwarm')
```



Seaborn 라이브러리 : 막대그림

막대그림: 도수표 + 제 2의 명목형 변수.

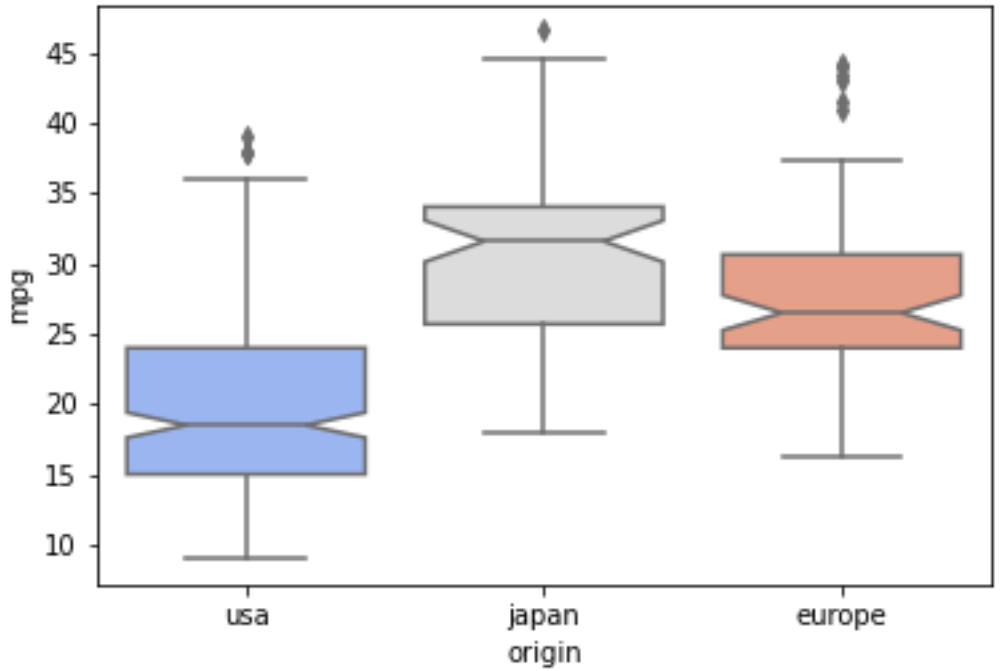
```
fig = sns.countplot(data=df, x='origin', hue='cylinders')
```



Seaborn 라이브러리 : 상자그림

상자그림: 세로 방향.

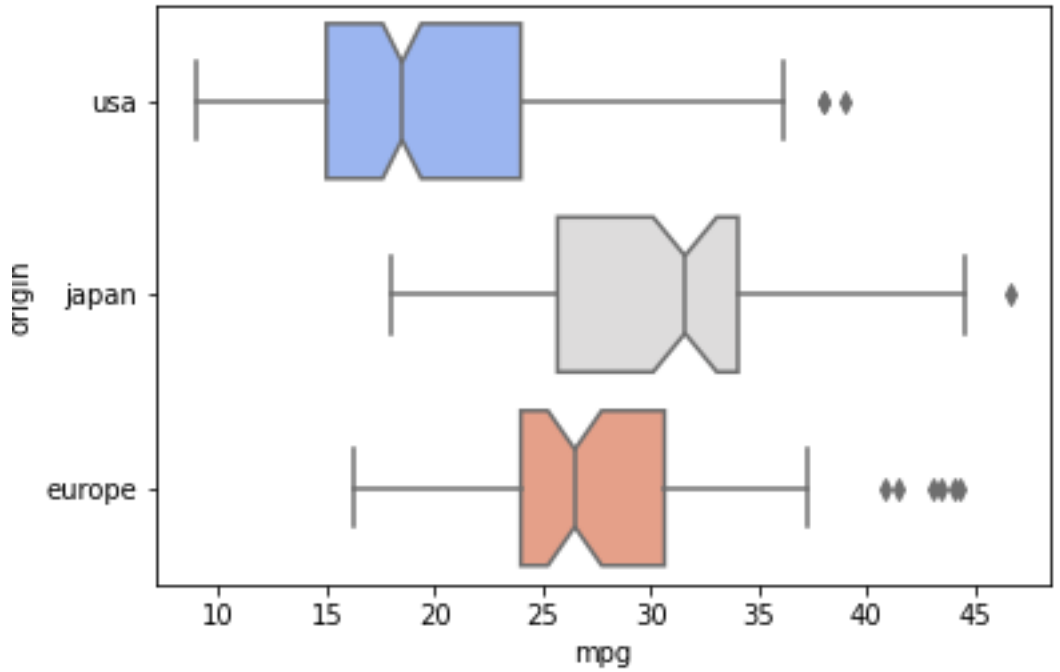
```
fig = sns.boxplot(data=df, x='origin', y='mpg', palette='coolwarm', notch=True)
```



Seaborn 라이브러리 : 상자그림

상자그림: 가로 방향.

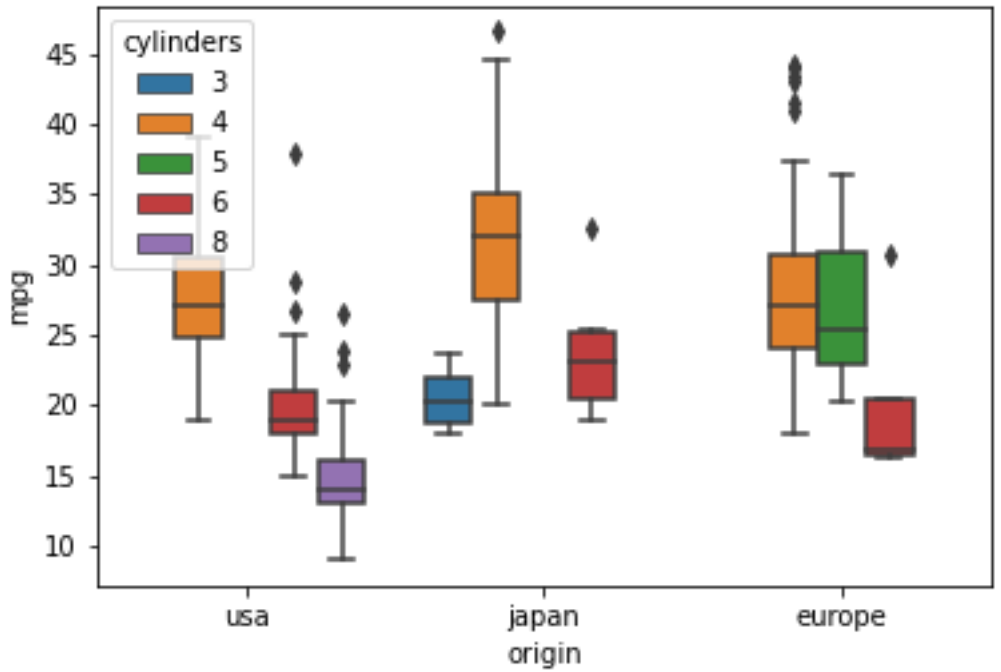
```
fig = sns.boxplot(data=df, x='mpg', y='origin', palette='coolwarm', notch=True)
```



Seaborn 라이브러리 : 상자그림

다중 상자그림: hue 인자 사용.

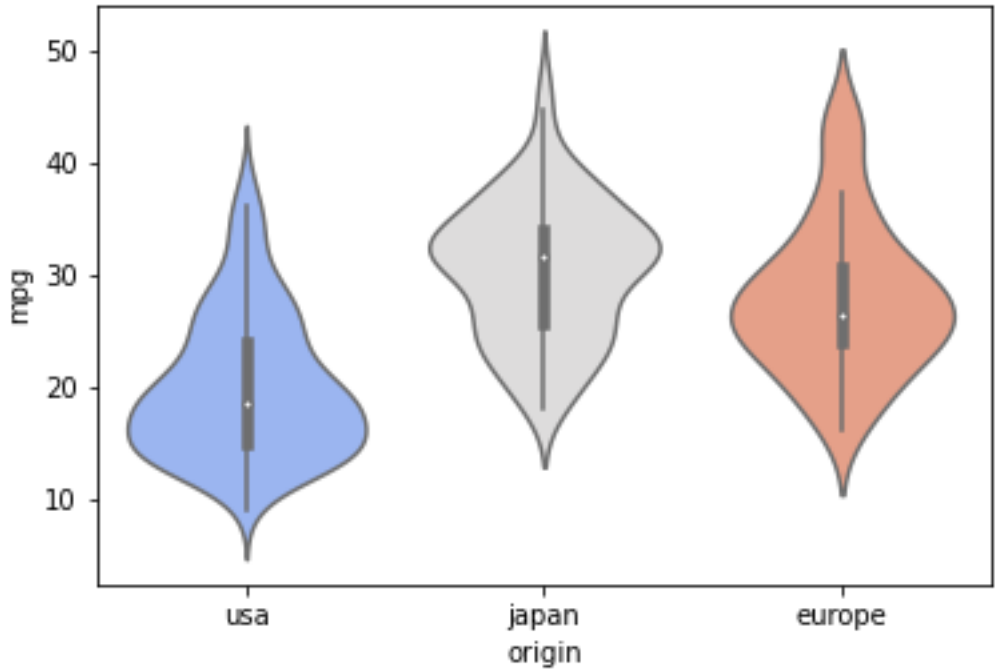
```
fig = sns.boxplot(data=df, x='origin', y='mpg', hue='cylinders')
```



Seaborn 라이브러리 : 바이올린

바이올린: 상자그림과 인자가 같음.

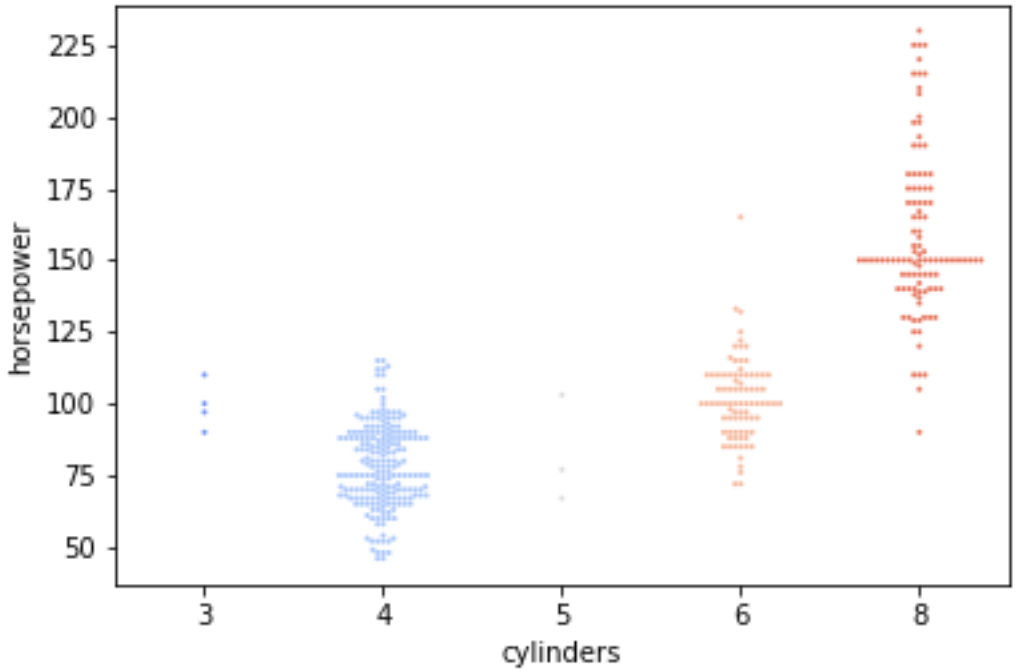
```
fig = sns.violinplot(data=df, x='origin', y='mpg', palette='coolwarm')
```



Seaborn 라이브러리 : Swarm

Swarm:

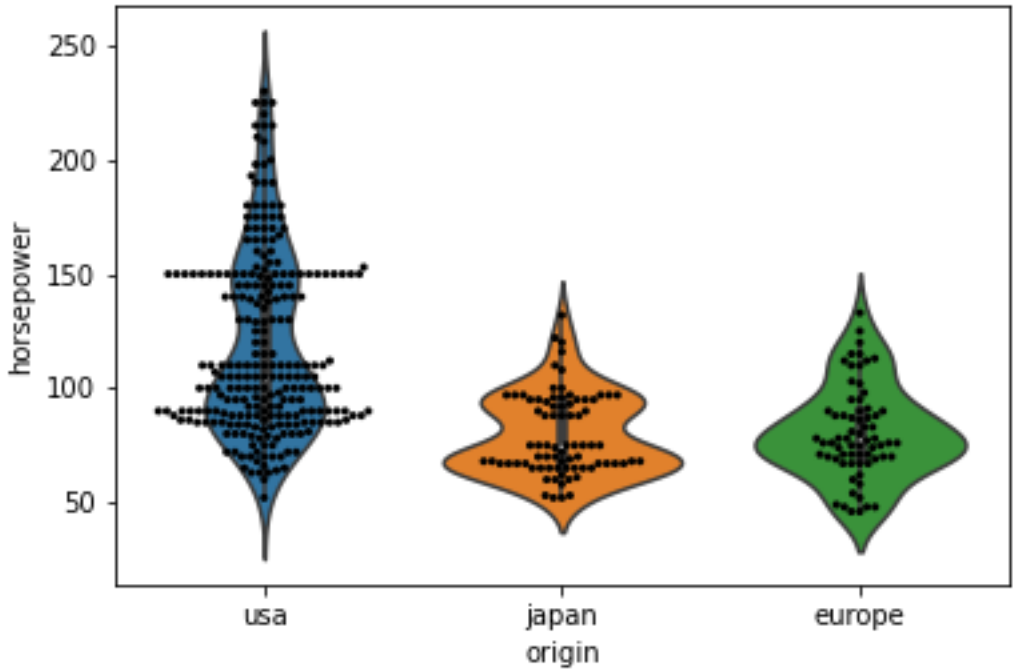
```
fig = sns.swarmplot(data=df, x='cylinders', y='horsepower', palette='coolwarm', size=2)
```



Seaborn 라이브러리 : 바이올린 + Swarm

바이올린 + Swarm:

```
sns.violinplot(data=df, x='origin', y='horsepower')
fig = sns.swarmplot(data=df, x='origin', y='horsepower', color='black', size=3)
```



실습 #0101

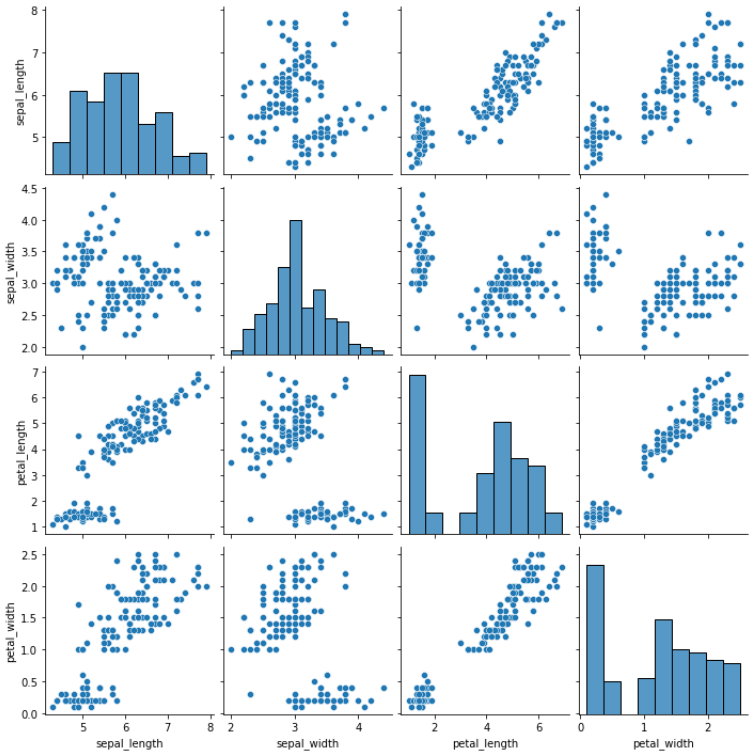
→ Seaborn 시각화 I. ←

→ 사용: **ex_0101.ipynb** ←

Seaborn 라이브러리 : 산점도 행렬

산점도 행렬:

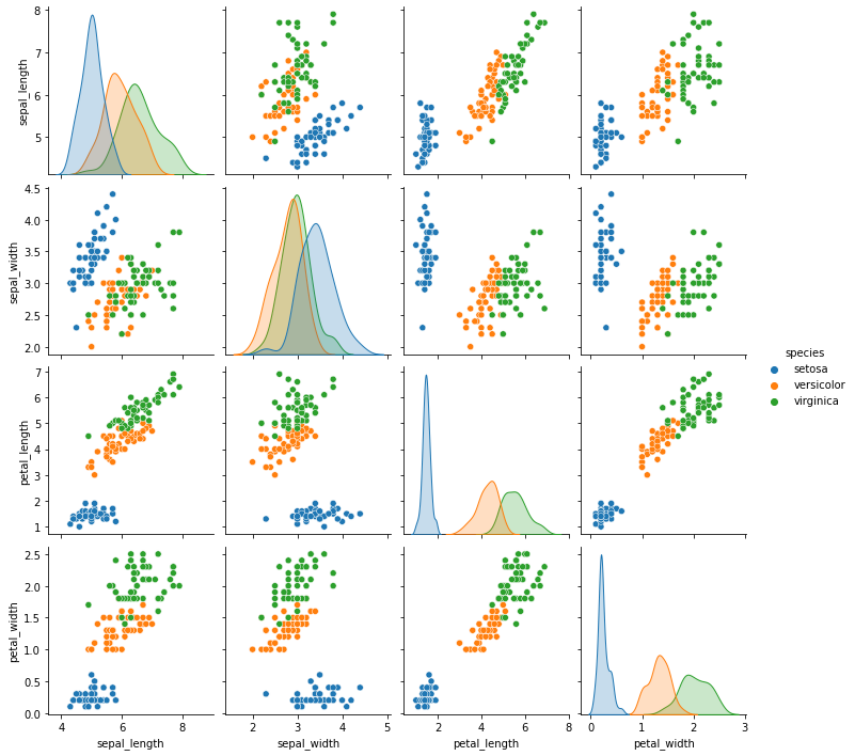
```
fig = sns.pairplot(data=df)
```



Seaborn 라이브러리 : 산점도 행렬

산점도 행렬: 유형을 색상으로 구별.

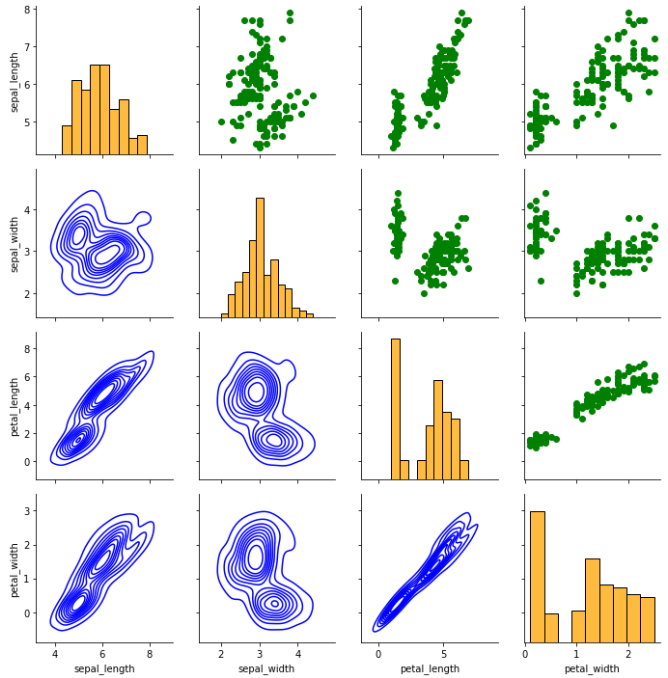
```
fig = sns.pairplot(data=df, hue='species' )
```



Seaborn 라이브러리 : 혼종 시각화 행렬

혼종 시각화 행렬:

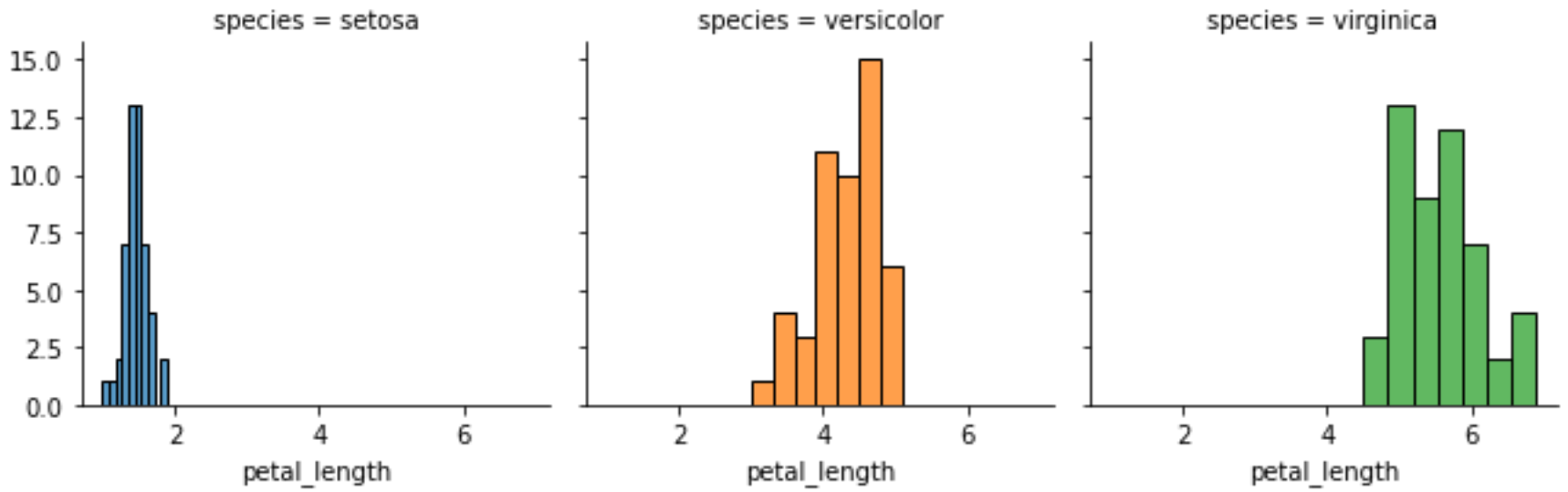
```
g = sns.PairGrid(data=df)
g.map_diag(sns.histplot, color='orange')          # 대각선 = 히스토그램.
g.map_upper(sns.scatterplot, color='green')       # 위 삼각 =  산점도.
fig = g.map_lower(sns.kdeplot, color='blue')      # 아래 삼각 = KDE.
```



Seaborn 라이브러리 : 다중 시각화

다중 시각화:

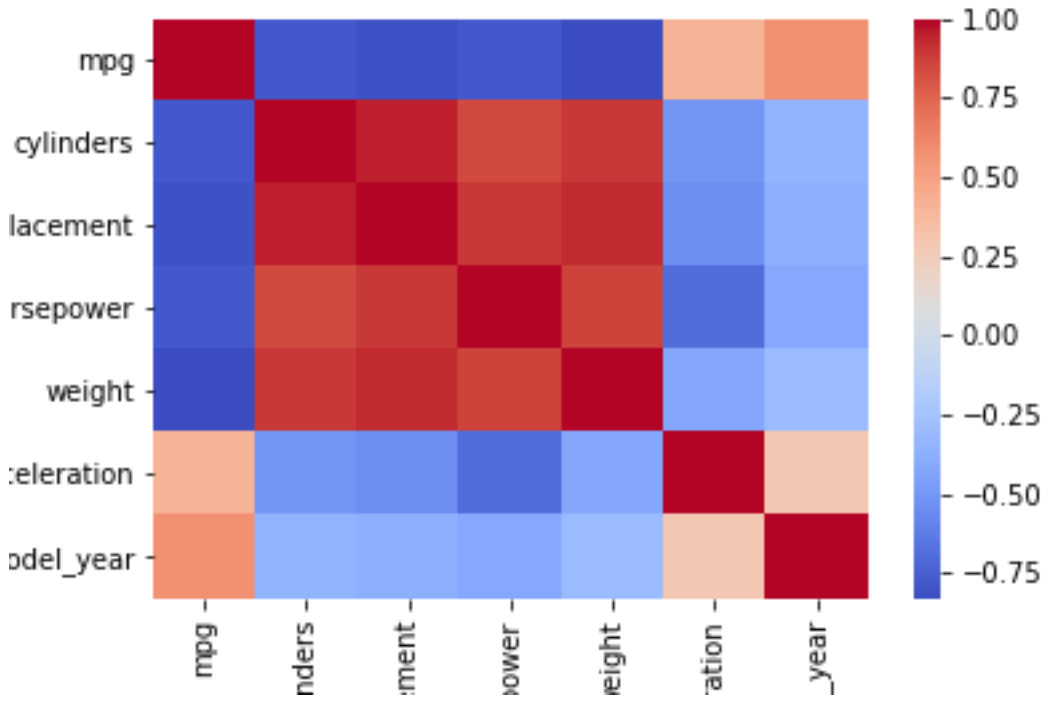
```
g = sns.FacetGrid(data=df, col='species', hue='species’)
fig = g.map(sns.histplot, 'petal_length')
```



Seaborn 라이브러리 : Heatmap

Heatmap:

```
fig = sns.heatmap(data=x, cmap='coolwarm')
```



실습 #0102

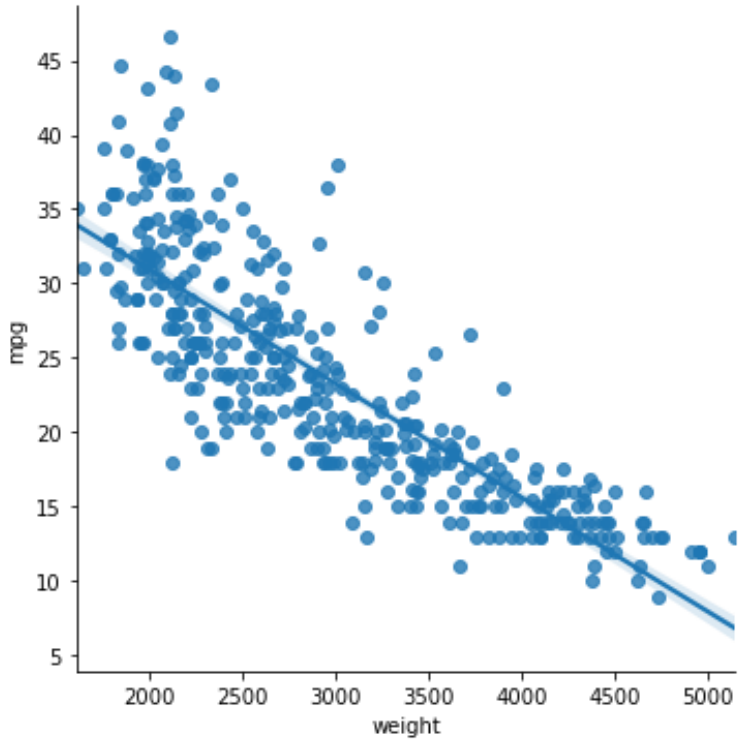
→ Seaborn 시각화 II. ←

→ 사용: **ex_0102.ipynb** ←

Seaborn 라이브러리 : 산점도 + 회귀선

산점도 + 회귀선:

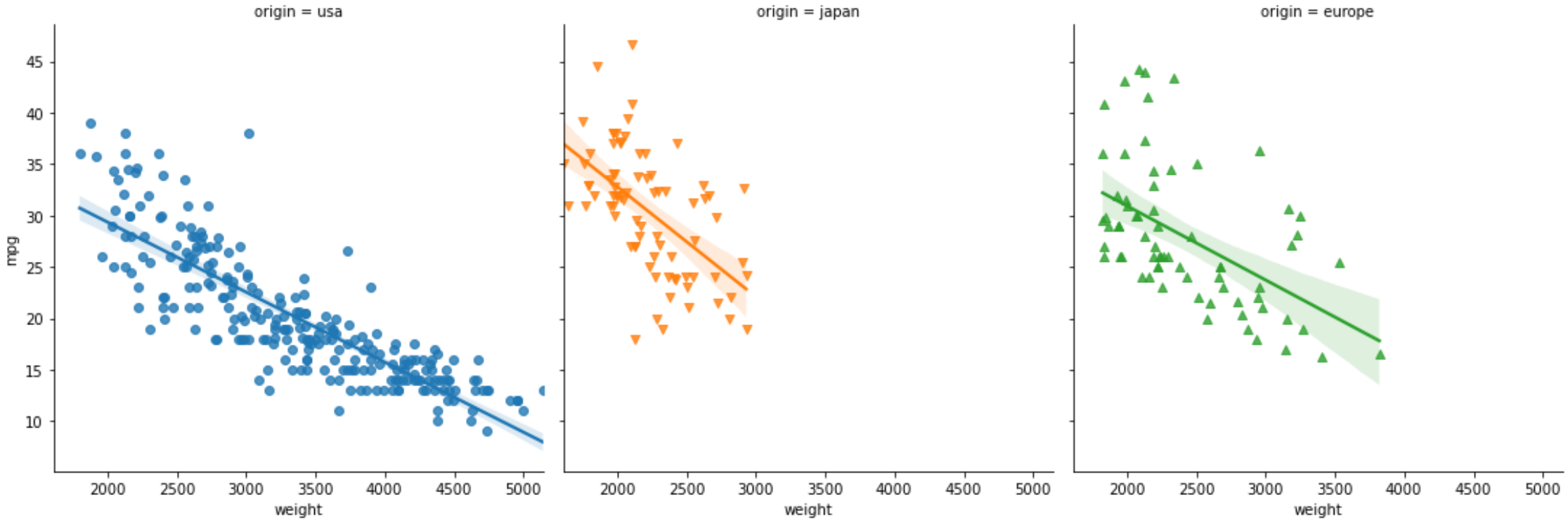
```
fig = sns.lmplot(data = df, x='weight', y='mpg')
```



Seaborn 라이브러리 : 산점도 + 회귀선

산점도 + 회귀선:

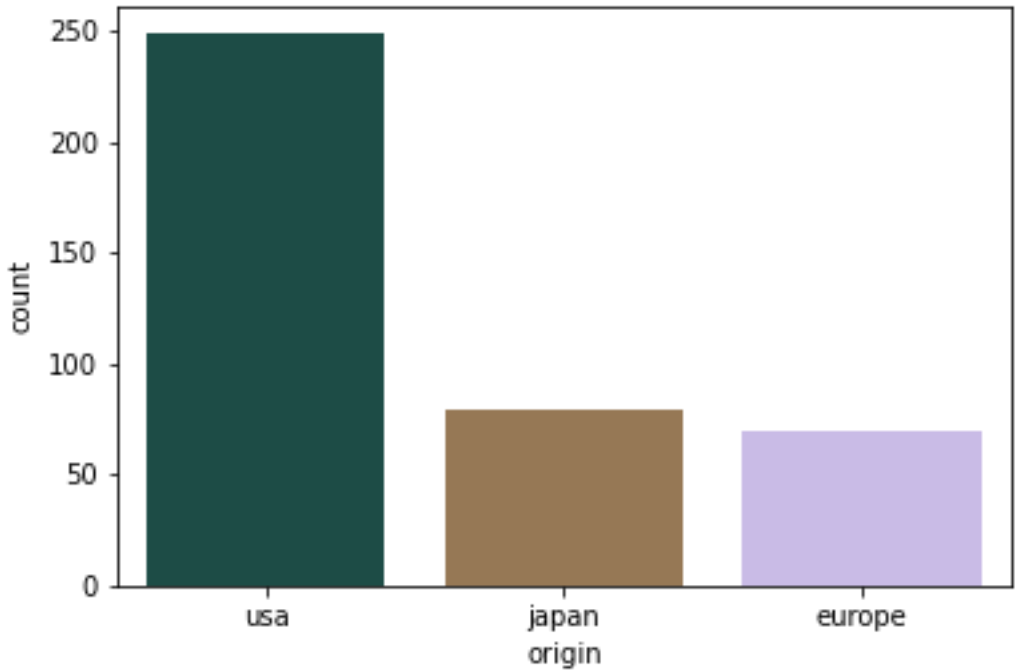
```
fig=sns.lmplot(data=df, x='weight', y='mpg', col='origin', hue='origin', markers=['o','v','^'])
```



Seaborn 라이브러리 : 컬러 Palette

컬러 Palette:

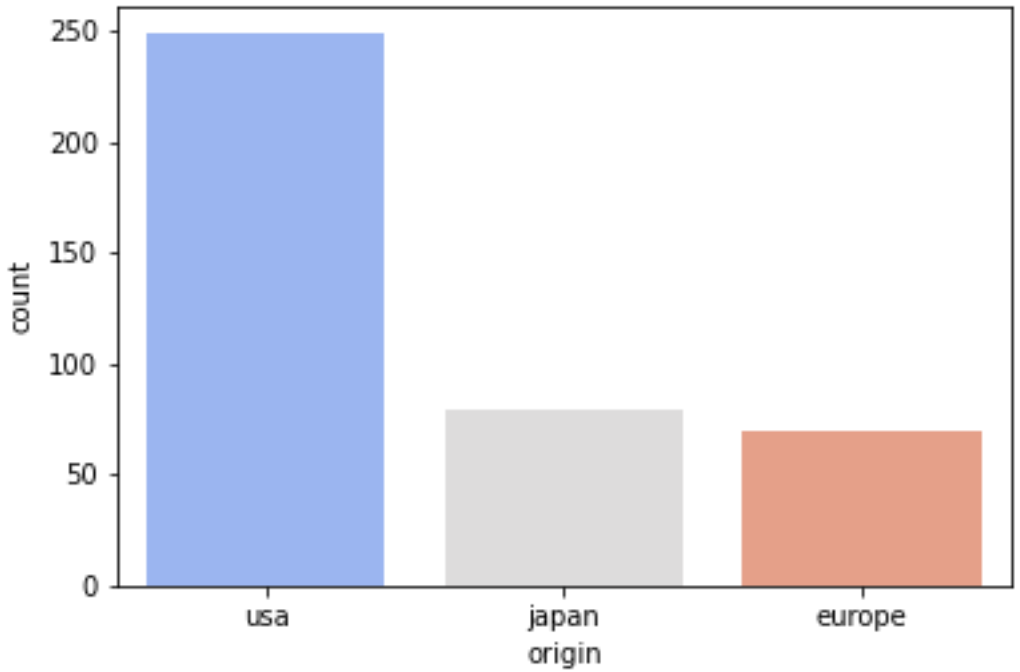
```
fig=sns.countplot(data=df, x='origin', palette='cubehelix')
```



Seaborn 라이브러리 : 컬러 Palette

컬러 Palette:

```
fig=sns.countplot(data=df, x='origin', palette='coolwarm')
```



실습 #0103

→ Seaborn 시각화 III. ←

→ 사용: **ex_0103.ipynb** ←

순서

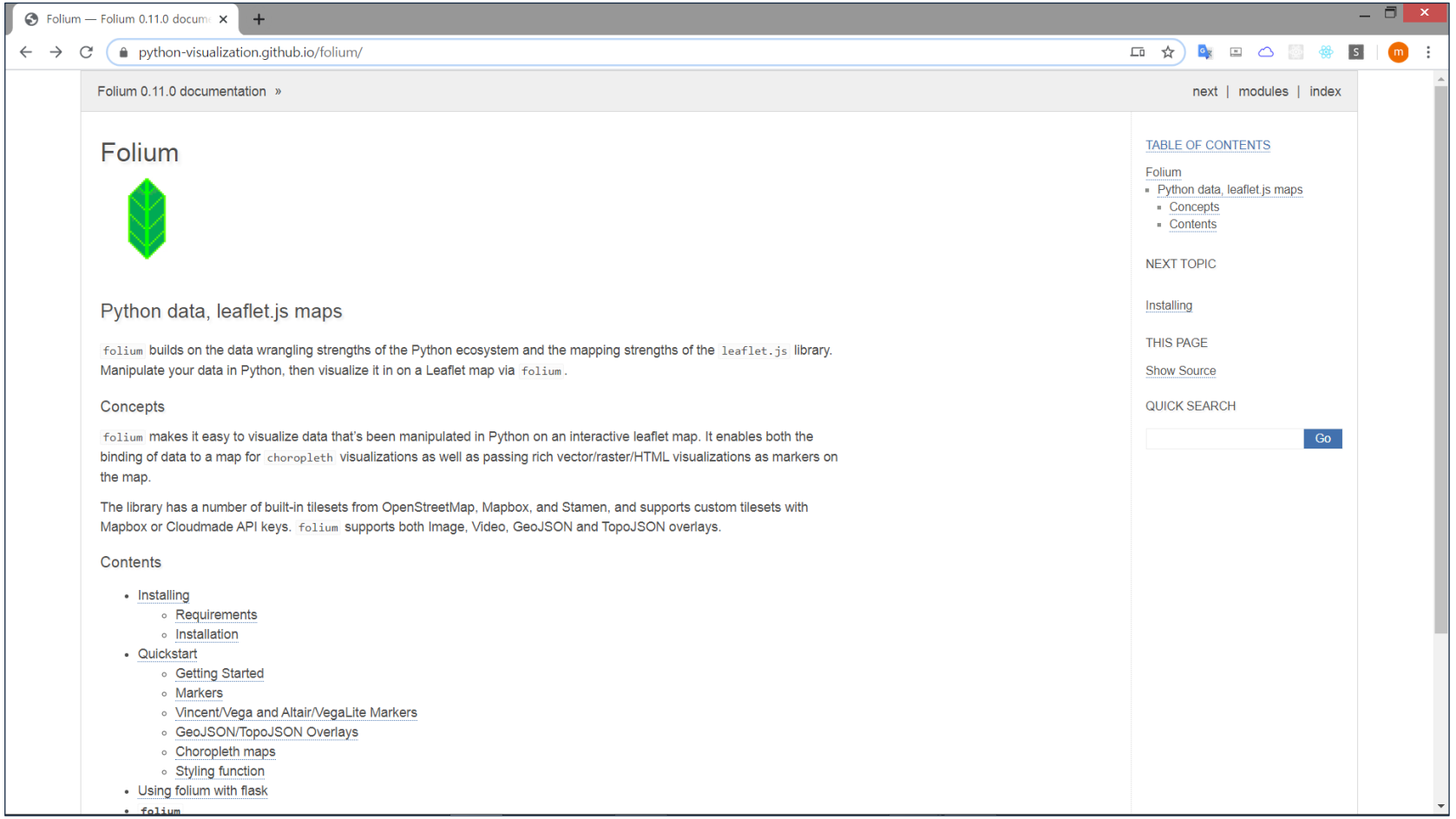
Folium 라이브러리 : 개요

Folium의 주요기능:

- 지리정보 (GIS) 시각화를 위한 패키지.
- JavaScript의 Leaflet.js 라이브러리의 Python 버전.
- Interactive한 시각화 제공.
- 원하는 위치, zoom, tiles 등을 적용한 map 생성.
- Map에 다양한 유형의 marker를 붙이고 customizing 할 수 있다.
- GeoJSON과 TopoJSON 지원.
- Choropleth map 지원.

Folium 라이브러리 : 개요

사이트: *https://python-visualization.github.io/folium/*



Folium 라이브러리 : Tiles

Map의 Tiles 속성:

- Map을 생성할 때 다양한 tiles 속성을 적용할 수 있다.

  ⇒ OpenStreetMap (기본형).

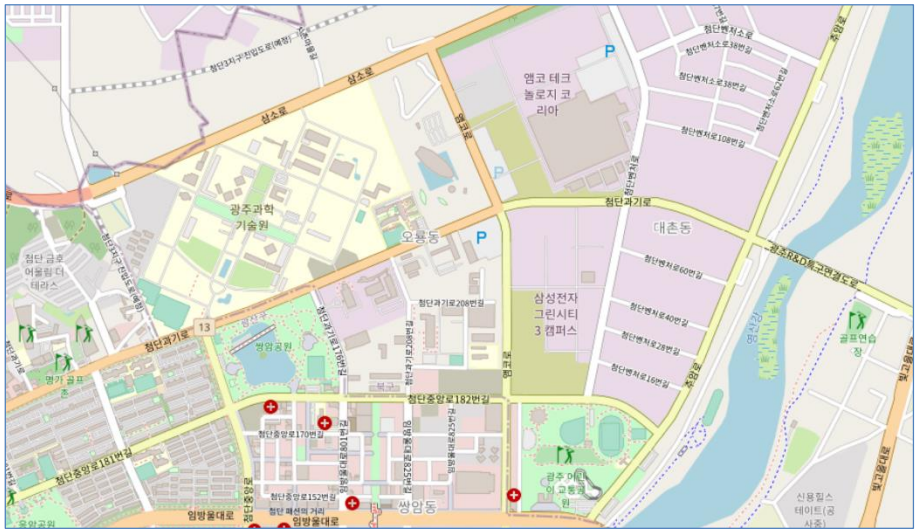
  ⇒ Stamen Terrain.

  ⇒ Stamen Toner.

  ⇒ Stamen Watercolor.

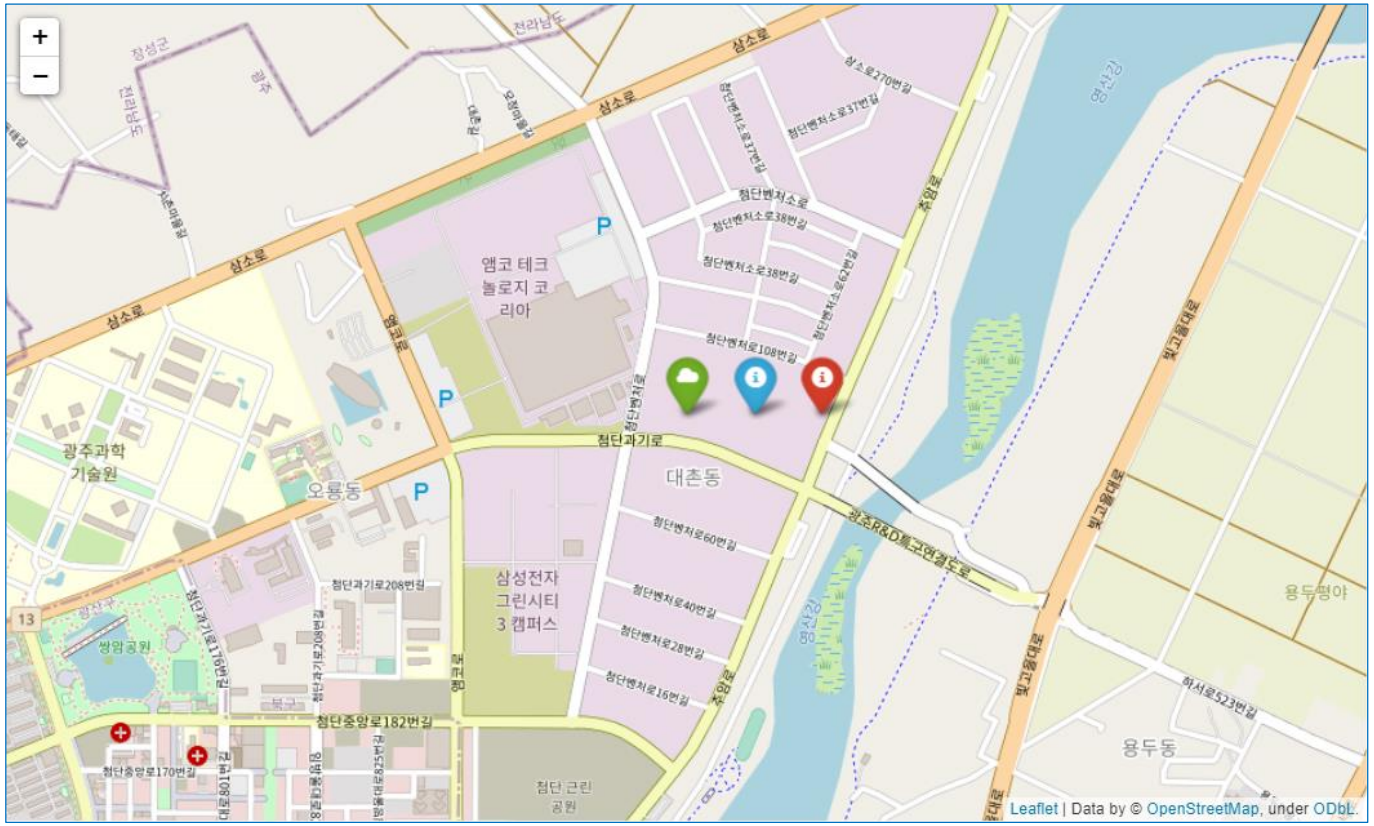
  ⇒ CartoDB Positron.

  ⇒ CartoDB Dark_matter.



Folium 라이브러리 : Marker

Map에 Marker 붙이기:



Folium 라이브러리 : Marker

Map에 Marker 붙이기:

- 다음과 같은 방법으로 map에 marker를 붙일 수 있다.

```
my_marker.add_to( my_map )        # 방법 1: marker가 map에 붙는다.

my_map.add_child( my_marker )     # 방법 2: map에서 marker를 가져다 붙인다.
```

- 다음과 같은 marker를 생성하기 위한 class가 있다.

⇒ Marker (기본형).

⇒ CircleMarker.

⇒ ClickForMarker.

⇒ LatLngPopup.

⇒ MarkerCluster.

⋮

Folium 라이브러리 : Icon

Marker에 Icon 입히기:

- Marker의 icon 인자에 Icon 객체를 대입해서 적용한다.

  folium.Marker(location=[*위도, 경도*], icon=folium.Icon(color='orange', icon = 'bookmark')).

- 다음과 같은 icon의 유형이 있다.

  ⇒ info-sign (기본형).

  ⇒ cloud.

  ⇒ flag.

  ⇒ flash.

  ⇒ bookmark.

  ⋮

실습 #0104

→ Folium 기초. ←

→ 사용: **ex_0104.ipynb** ←

Folium 라이브러리 : GeoJSON

GeoJSON에 대해서:

- JSON (*) 으로 위치 데이터와 속성 데이터를 저장하는 공개 표준 형식이다.
- 다음과 같은 특장점이 있다.

  ⇒ 문법이 간결하며 유연하다 (XML 형식과 비교해서).

  ⇒ JSON이므로 다양한 프로그래밍 언어에서 객체화 할 수 있다.

  ⇒ 세계 지구 좌표 시스템 (WGS 84)가 권장하는 “[경도, 위도]”와 같은 위치 표기법을 지원한다.

(*) JSON = **J**ava**S**cript **O**bject **N**otation.

Folium 라이브러리 : GeoJSON

GeoJSON에 대해서:

- Feature 들의 집합 “FeatureCollection” 으로 이루어져 있다.
- 하나의 feature는 다음과 같이 구성되어 있다.

```
{
"type": "Feature",
"geometry": {
  "type":  Geometry의 유형 ,
  "coordinates": [경도, 위도] 리스트
},
"properties": {
  속성 이름: 속성의 값
 }
}
```

Folium 라이브러리 : GeoJSON

GeoJSON에 대해서:

- Feature 들의 집합 “FeatureCollection” 으로 이루어져 있다.
- 하나의 feature는 다음과 같이 구성되어 있다.

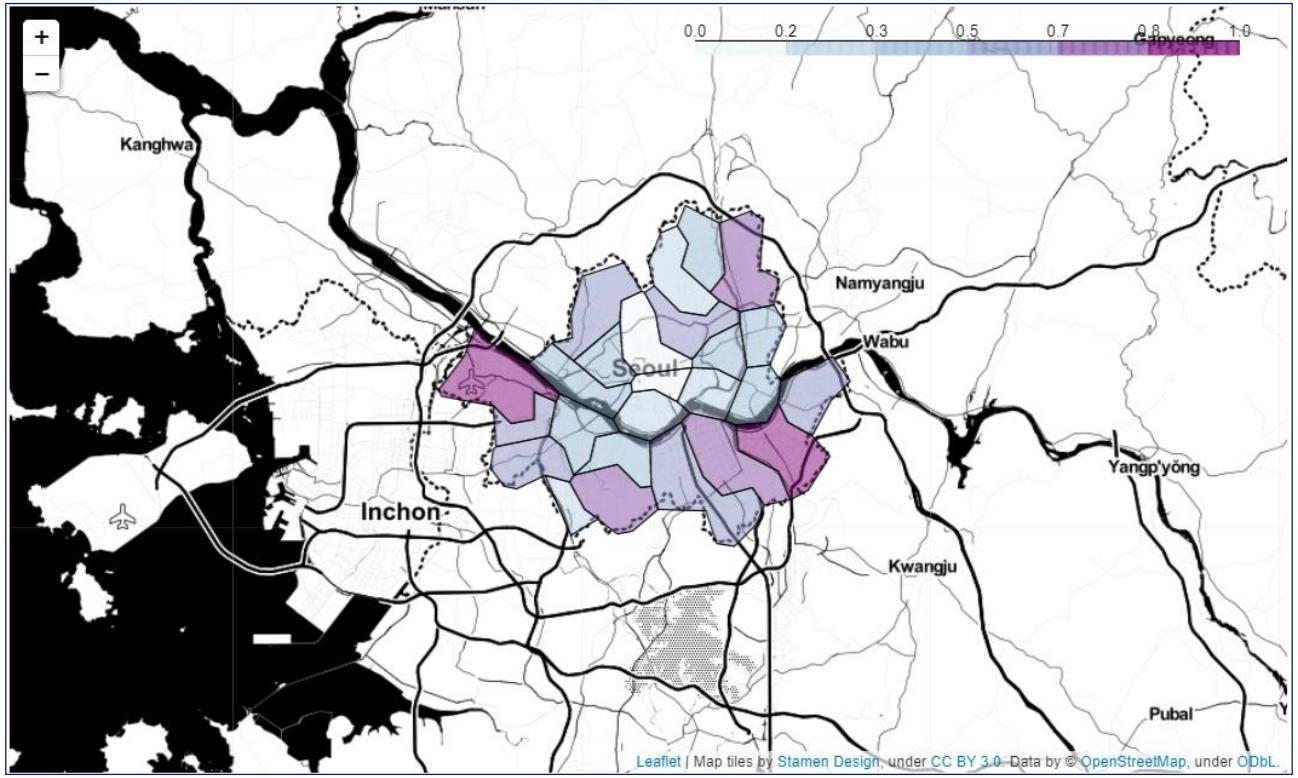
```
{
"type": "Feature",
"geometry": {
  "type":  Geometry의 유형 ,
  "coordinates": [경도, 위도] 리스트
},
"properties": {
  속성 이름: 속성의 값
 }
}
```

기본 유형: Point, LineString, Polygon.

다중 유형: MultiPoint, MultiLineString, MultiPolygon.

Folium 라이브러리 : Choropleth

GeoJSON 활용: Choropleth.



⇒ Area 경계선을 그려서 Shading을 할 수 있다.

실습 #0105

→ Folium의 Choropleth Map. ←

→ 사용: **ex_0105a.ipynb** , **ex_0105b.ipynb** ←

순서

Streamlit 라이브러리 : 개요

Streamlit에 대해서:

- Python의 data dashboarding 라이브러리.
- 2020년 이후 사용자가 급격히 증가하고 있다!
- 다음과 같은 특장점이 있다.

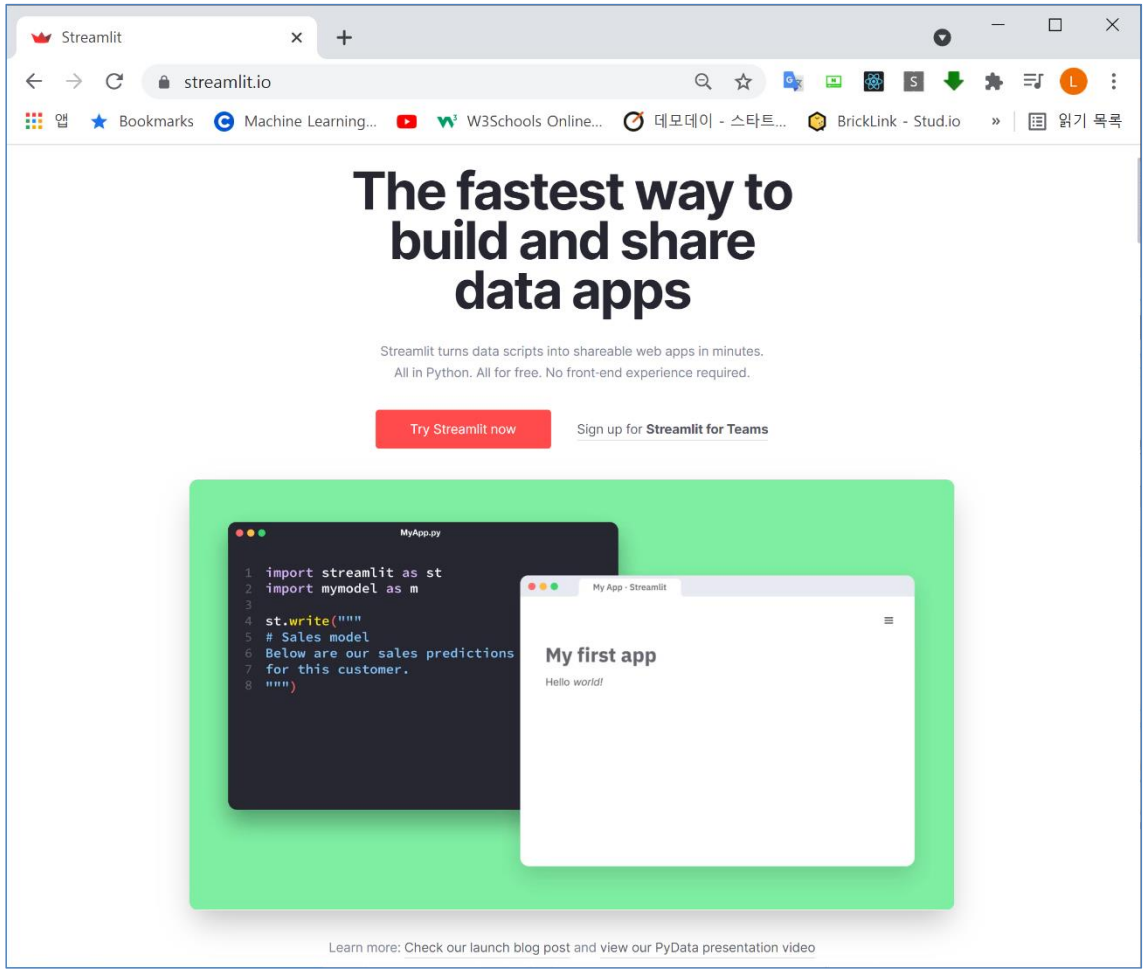
  ⇒ 설치가 간단하고 배우기 쉽다.

  ⇒ 사용하기 간편한 markdown과 widget으로 시각화 App을 작성할 수 있다.

  ⇒ Backend 프로그래밍이 필요 없으며 배포가 쉽다.

Streamlit 라이브러리 : 개요

사이트: *https://streamlit.io/*



Streamlit 라이브러리 : 개요

Streamlit 설치와 앱 실행:

- Command line에서 직접 명령어를 실행할 수 있는 개발환경이 필요하다. → 예: VSCode
- 다음과 같이 Streamlit을 Python 가상환경에 설치하고 (*), 데모 앱을 런칭할 수 있다.

  pip install streamlit

  streamlit hello
- 텍스트 문서로 작성된 .py 프로그램 파일을 다음과 같이 실행할 수 있다.

  streamlit run *파일이름.py*

(*) 가상환경을 설정하고 필요한 라이브러리들을 일괄적으로 설치하는 방법은 **부록**을 참고한다.

Streamlit 라이브러리 : Text 출력

Text 출력 함수: *https://docs.streamlit.io/en/stable/api.html*

함수	기능
st.text(문자열)	작은 크기의 text 출력. Markdown 인식 못함.
st.markdown(문자열)	Markdown을 컴파일해서 출력.
st.write(문자열 또는 객체)	Markdown 또는 Python의 객체 출력.
st.title(문자열)	타이틀 대. Markdown 인식함.
st.header(문자열)	타이틀 중. Markdown 인식함.
st.subheader(문자열)	타이틀 소. Markdown 인식함.
st.caption(문자열)	작은 크기의 caption. Markdown 인식함.

⇒ “st”는 “streamlit” 라이브러리의 약칭.

Streamlit 라이브러리 : Chart 출력

Chart 출력 함수: *https://docs.streamlit.io/en/stable/api.html*

함수	기능
st.line_chart(data)	꺾은선 그래프 출력.
st.area_chart(data)	Area chart 출력.
st.bar_chart(data)	막대 그래프 출력.
st.pyplot(figure 객체)	Matplotlib의 figure 객체 출력.

⇒ 함수 일부만 간추림.

Streamlit 라이브러리 : Widget

Widget 함수: *https://docs.streamlit.io/en/stable/api.html*

함수	기능
st.button(문자열)	Button의 클릭 감지.
st.checkbox(문자열)	Checkbox 한 개.
st.radio(문자열 , 리스트)	리스트 아이템 바탕으로 radio button 그룹 생성.
st.selectbox(문자열 , 리스트)	리스트 아이템 중 한가지를 drop down 선택.
st.multiselect(문자열 , 리스트)	리스트 아이템 중 다중 선택.
st.slider(문자열, 최소값, 최대값, 초기값)	Slider를 통해서 수치 값 입력.
st.number_input(문자열)	수치 자료 입력 창.
st.text_input(문자열)	문자열 자료 입력 창.

⇒ 함수 일부만 간추림.

Streamlit 라이브러리 : 기타

기타 유용한 기능: *https://docs.streamlit.io/en/stable/api.html*

- 다음과 같이 sidebar 안에 widget을 삽입 할 수 있다.

  x = st.sidebar.*widget*

- 다음과 같은 decorator를 사용해서 페이지가 reload 될 때 함수의 실행을 향상 시킬 수 있다.

  @st.cache

- 다음과 같이 실행 상태를 출력해 주는 함수가 있다 (일부만 간추림).

함수	기능
st.error(문자열)	에러 메시지 출력.
st.warning(문자열)	경고 메시지 출력.
st.info(문자열)	Info 메시지 출력.
st.success(문자열)	성공 메시지 출력.

실습 #0106

→ Streamlit의 기능을 알아보고 간단한 시각화/ML App을 만들어 본다. ←

→ 사용: **ex_0106.ipynb** , **ex_0106a.py** ~ **ex_0106e.py** ←

순서

부록 : Conda 가상환경

Conda와 가상환경 관리:

- Conda는 Anaconda prompt에서 실행할 수 있는 가상환경 관리자이다.

  ⇒ 가상환경을 통해서 프로젝트 별 개발 환경을 효율적으로 구성하고 관리할 수 있다.

목적	실행
버전 확인	conda --version
클린한 상태의 가상환경 생성	conda create -n <가상환경 이름> python=<파이썬 버전>
Anaconda 패키지 포함한 가상환경 생성	conda create -n <가상환경 이름> anaconda python=<파이썬 버전>
가상환경 활성화	conda activate <가상환경 이름> 또는 activate <가상환경 이름>
설치된 패키지 목록 확인	conda list
가상환경 목록 보기. "*" = 현재 환경.	conda env list 또는 conda info --envs
Base 가상환경으로 나가기	conda deactivate
가상환경 제거	conda env remove -n <가상환경 이름>
패키지 설치와 제거	conda install <패키지 이름> 와 conda uninstall <패키지 이름>

부록 : 개발환경 설치

개발환경 설치하는 과정:

1). 다음과 같이 가상환경을 만들고 들어간다 (option).

```
(base) C:□Users□USER> conda create –n likelion python=3.8.8
(base) C:□Users□USER> conda activate likelion
(likelion) C:□Users □USER>
```

2). 가상환경에서 다음과 같이 필요한 라이브러리를 일괄적으로 설치할 수 있다.

```
(likelion) C:□Users □USER> pip install –r requirements.txt
```

3). 다시 **base** 환경으로 나오려면 다음을 실행한다.

```
(likelion) C:□Users □USER> conda deactivate
```

4). 가상환경을 삭제하려면 다음을 실행한다.

```
(base) C:□Users □USER> conda env remove –n likelion
```

모듈 #1 : 끝

문의:

sychang1@gmail.com